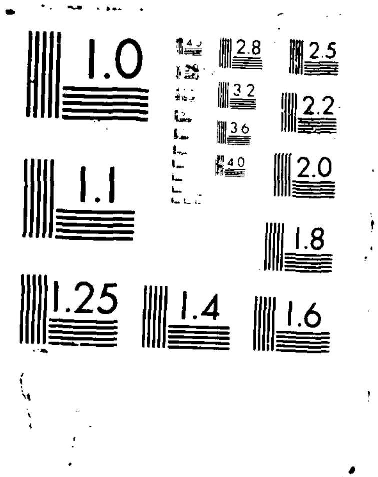
AD-A190 571 THE MODELING SIMULATION AND COMPARISON OF 1/2
INTERCONNECTION NETWORKS FOR PA..(U) AIR FORCE INST OF
TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI..
UNCLASSIFIED R A RAINES DEC 87 AFIT/GCE/ENG/87D-8 F/G 25/5 NL


1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

THE MODELING, SIMULATION AND COMPARISON INTERCONNECTION NETWORKS FOR PARALLEL PROCESSING

THESIS

Richard A. Raines
Second Lieutenant, USAF

AFIT/GCE/ENG/87D-8

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCE/ENG/87D-8

THE MODELING, SIMULATION AND COMPARISON OF INTERCONNECTION NETWORKS FOR PARALLEL PROCESSING

THESIS

Richard A. Raines
Second Lieutenant, USAF

AFIT/GCE/ENG/87D-8

AFIT/GCE/ENG/87D-8

THE MODELING, SIMULATION AND COMPARISON OF INTERCONNECTION NETWORKS FOR PARALLEL PROCESSING

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Computer Engineering

Richard A. Raines, B.S.E.E.
Second Lieutenant, USAF

December 1987

Approved for public release; distribution unlimited

Acknowledgments

The results obtained from this thesis culminates a nine month effort that has seen both times of frustration and times of elation. Though the front of this thesis reflects the name of a single individual, this effort could not have been completed without the assistance from others. Foremost, this thesis investigation was funded by the Strategic Defense Initiative Organization which also provided funds to purchase the Sun workstations used to document this investigation.

In the course of this investigation, several individuals lent their time, guidance, patience, and understanding to me. I wish to express my gratitude to my thesis advisor, Captain N. J. Davis IV, first for giving me guidance when I needed it and second, for enduring the reading and numerous iterations of writings needed to obtain the final product. I also wish to thank my committee members, LTC Albert B. Garcia and Captain Wade H. Shaw for their time and cooperation in this investigation. Special thanks are in order to Captain Shaw, who always answered my simulation questions and provided invaluable assistance. A word of thanks is also due to Captain Mark Strovink, AFIT/SI for providing the means to perform the simulations of this investigation.

Finally, I would like to add a little more than the customary expression of appreciation to my wife Helen for her encouragement, understanding, and love throughout this effort. We made it!

Richard A. Raines

Table of Contents

List of Figures

List of Tables

AFIT/GCE/ENG/87D-8

Abstract

This thesis extends existing modeling, analysis and comparison of interconnection networks for parallel processing systems. Simulation models are developed for the multistage cube network, the single stage cube network (hypercube), and the Illiac IV mesh-type network. They are then used to provide a comparison of three classes of interconnection networks which, until now, has not been performed. These models, implemented using a commercially packaged simulation language provide for compact source code and ease of readability. The networks are modeled under a common set of operating assumptions and system environment. This allows for accurate comparisons of average network packet delays and memory requirements necessary to physically implement the chosen network at a given network operating load. It is concluded that, for the network sizes and operating conditions established, the multistage cube network performs better at a lower hardware cost than do the single stage cube and mesh networks. As a result, the designer of a parallel processing system is given additional insight for choosing an interconnection network which best suites the application needs. This thesis investigation is summarized in [RaD88a], and [RaD88b].

THE MODELING, SIMULATION AND COMPARISON OF INTERCONNECTION NETWORKS FOR PARALLEL PROCESSING

1. Introduction

1.1 Background

Since the ad ·ent of the modern computer, computer architects have continuously attempted, and in most cases succeeded, in designing and implementing faster, more powerful systems. Due to design innovations and technological advances, computer systems have rapidly expanded and diversified from the uniprocessor, von Neumann architecture [Von46]. *In the relatively short period of forty years*, computing capabilities have increased from performing simple mathematical functions to those of performing complex numerical and artificial intelligence applications. Traditionally, the computational power needs of society have exceeded the processing power of contemporary computer systems. For this reason, ongoing research is investigating and proposing possible ways of meeting or exceeding the processing needs of society.

Processing tasks such as weather forecasting, ballistic missile defense, image processing, air-traffic control, pattern and speech recognition, medical diagnosis, and robotic vision are issues that concern society. These tasks, due to their nature, require the computational speeds of the host machine to be near or at real-time speeds.

Due to the computational complexity of the algorithms required to implement the above named tasks, real-time processing is, in some cases, not feasible in a uniprocessor environment. The real-time constraints imposed by these tasks require the computer architect to look for ways of expanding from the uniprocessor von Neumann architecture to accomplish these tasks.

In many cases, the uniprocessor von Neumann architectures lack the ability to perform real-time applications. This is due to their sequential mode of operation and largely to the computational complexity of the algorithms executed. In the von Neumann machine, instructions must be executed in a sequential manner. Design techniques which incorporate overlapping and pipelining of instructions allow for execution times to be greatly reduced. These techniques, coupled with the technological advances in very large scale integrated (VLSI) circuit design have brought single processor computational speeds to near the speeds necessary for real-time applications. To achieve the real-time processing capabilities, concurrent processing must take place. Parallel processing systems, comprised of many cooperating processors, have been developed in effort to achieve the real-time processing capabilities.

One of the major concerns in designing parallel processing systems lies in determining the interconnection scheme of the multiple processors. The designer must consider the application and the number of processors to be used when choosing an interconnection network. Numerous interconnection networks have been implemented to link multiple processors to achieve a parallel computing environment [Fen81]. The goal of the interconnection network design is to have the communication time between processors be substantially less than the processing time required by an individual processor to execute an instruction or set of instructions.

Further design issues exist once an interconnection network topology has been chosen for study. First, the switching methodology must be determined. The switching methodology determines the way in which a message is to be routed through the switching elements of the network. Four methods exist: circuit switching (dedicated physical transmission paths are set up and held until completion of transmission); packet switching (transmission paths are dynamically allocated and released upon the receipt and passage of the data packet); virtual cut through (packet headers are examined and forwarded to the next appropriate channel before receipt of entire packet) [KeK79]; and wormhole routing, a derivative of virtual cut through (blocked packets remain in the network instead of being buffered as in virtual cut through) [Dal86].

The control strategy of the network switching elements is another design issue that must be examined. Two types of control strategies exist; control which is distributed at each switching element and a centralized control which is used for all switching elements. Trade-offs exists for using either of the two control strategies. Chapters 2 and 3 provide a more detailed discussion of the control strategies as well as switching methodologies and interconnection network design and implementations.

As the number of processors in a parallel system has grown, so has the complexity of analytical approaches to modeling these systems. In most cases, it is no longer feasible, and in few cases possible, to determine the expected performance of a multiprocessor design using only mathematical techniques. As a result, many designers of multiprocessor systems have turned to modeling their systems through computer simulations. Computer simulation can provide a low-cost and time-efficient method for systems modeling. Simulation can provide supplemental information which can be used to validate mathematical models when available. Through simulation, it is possible to compare the performance of dissimilar designs and determine the design best suited for a particular application.

1.2 Research Goals

While many research efforts have examined the performance of particular networks under differing environments [DaS86, DiJ81, KrS83, Law75, Pat81, Pea77], few have performed comparisons of dissimilar interconnection networks under the same environment [AbP86, Dal86]. Of the research efforts which have performed interconnection network comparisons for the same operating conditions, their scope has been limited in the number of differing networks compared [AbP86, Dal86] and in the sizes of the switching elements which connect the multiple processors [AbP86]. For this reason, the main objective of this thesis investigation is to expand upon the performance comparison base by examining three types of interconnection networks: the single stage cube network, the multistage cube network and the Illiac IV mesh network. This performance comparison is made for network sizes capable of supporting 64 to 1024 autonomous processors. This range is chosen due to the technological limitations which presently exist in supporting complex processors. The physical structure and interconnection functions of these three interconnection networks are discussed in detail in Chapter 2.

One of the figures of merit that this investigation concentrates on is the average delay incurred by a message as it traverses the network. By comparing the average delay experienced by messages entered into the networks, for various network loading, a determination of the desirability of one network over another can be made. Added information in determining the performance of the three networks is gained through the knowledge of the maximum queue lengths associated with each network for a given network loading. This gives the designer insight into the cost of constructing a network if the average delay is the most important performance parameter considered.

1.3 Summary

This chapter has presented an overview of the processing restrictions incurred by using the traditional von Neumann architecture. Methods for overcoming these restrictions can be realized by using parallel processing techniques. One of the underlying problems associated with parallel systems is in the determination of how the multitude of processors will be connected to one another. This investigation examines three topologies of interconnection networks.

In Chapter 2, an overview of parallel processing systems is presented. A detailed discussion is presented on the classification of parallel systems. This discussion is followed by an examination of interconnection networks, the characteristics associated with message transmission and the control methodologies. A brief overview of contemporary parallel systems is presented with relation to their interconnection network, switching methodology, and control implementations.

Chapter 3 examines previous performance modeling and analysis research that has been performed. These studies examine both the analytic and the simulation modeling of interconnection networks. A discussion of the modeling and analysis of the switching methodologies of the network is presented, comparing circuit switching. and packet switching. The latter sections of Chapter 3 discuss the present state of network modeling and analysis. These sections examine the comparison of dissimilar network topologies.

Chapter 4 presents the methodology applied to solving this investigation. The simulation methodology, validation, and performance analysis of the three interconnection network models are presented in Chapter 5. Conclusions and recommendations for future research are presented in Chapter 6.

2. Parallel Processing Systems Overview

2.1 Introduction

In this chapter, the underlying characteristics of parallel processing systems will be reviewed. The basic understanding of these characteristics is essential to the comprehension of the contemporary concurrent processing systems and the architectural problems which they face. In Section 2.2, parallel processing systems classification methodologies will be presented. Section 2.3 will review the major classes of interconnection networks. Contemporary parallel processing systems will be discussed in Section 2.4 with relation to the interconnection networks discussed in Section 2.3.

2.2 Parallel Processing Systems Classification Methodologies

The ability to accurately classify computer systems at the systems level is a problem that has plagued computer architects since the inception of the von Neumann machine. Three taxonomies have been recognized as viable tools for use in reducing this problem.

2.2.1 Flynn's taxonomy In 1966, Michael Flynn [Fly66] proposed a method for classifying computer systems based on the number of instruction and data *streams* associated with the system. The term *stream* is used to denote the sequence of items (instructions or data) that are either executed or operated upon by a processor contained in the machine. From this concept of streams, Flynn proposed that a machine could be classified into one of four categories. These categories are as follows:

- Single Instruction stream - Single Data stream (SISD)
- Single Instruction stream - Multiple Data stream (SIMD)

- Multiple Instruction stream - Single Data stream (MISD)
- Multiple Instruction stream - Multiple Data stream (MIMD)

Figure 2.1 represents the four categories of Flynn's taxonomy.

The SISD machine represents the traditional von Neumann architecture. This machine is characterized by a single processor which uses a single instruction stream and a single data stream for its operation. The MISD machine is characterized by multiple instructions streams, supplied by multiple processors, which operate on a single data stream. In theory, a MISD machine's multiple processors operate concurrently on a single stream of data. At present, no true MISD machines exist [HwB84]. Both the SIMD and MIMD machines are classified as parallel processing systems. A SIMD machine is characterized by a single instruction stream which is spawned off to multiple processors, each of which retains its own data streams. The single instruction stream allows for a "lock-step" (sequential) instruction execution. The MIMD machine is one whose characteristics are truly parallel. An instruction stream is associated with each of the multiple processors in the system. This allows for the concurrent operation and execution of instructions.

2.2.2 Feng's Taxonomy T.Y. Feng's taxonomy [Fen72] attempts to compare computer systems by computing their degree of parallelism. From the results of these computations, Feng proposes that the processing power of a system can be quantified. The *degree of parallelism* represents the maximum number of bits per unit time that the system can process. To describe the measure of parallelism, Feng uses the ordered pair (n, m), where n is the processor word length and m is the system bit-slice length. Thus, systems can be classified as word serial/parallel ($n = 1$ or $n > 1$) and bit serial/parallel ($m = 1$ or $m > 1$). By computing the product of n and m, the degree of parallelism of the system can be used for performance comparisons of differing architectures.

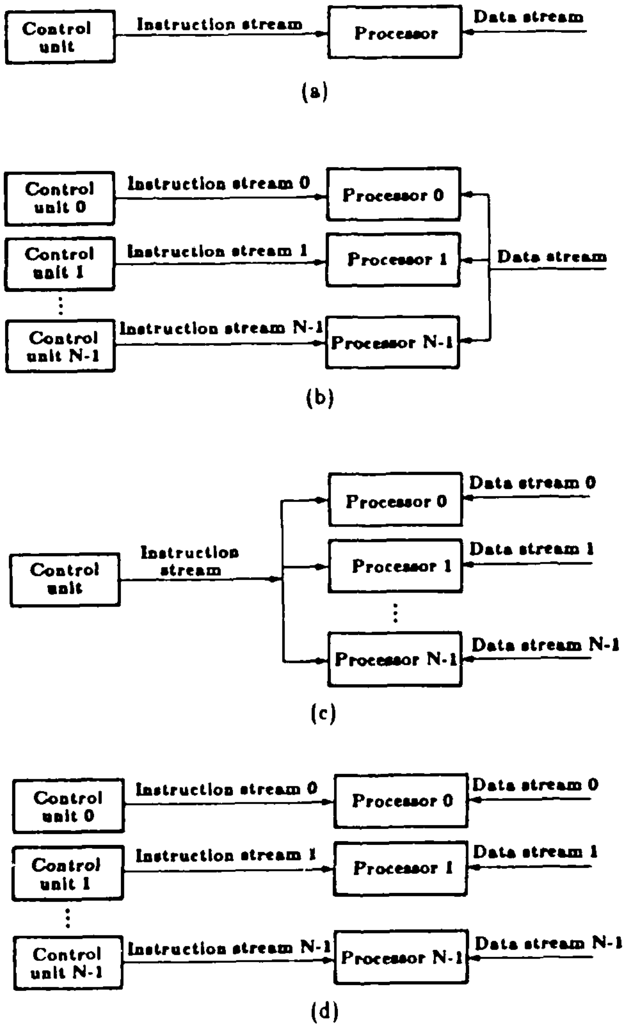


Figure 2.1. Flynn's classification taxonomy. (a) SISD; (b) MISD; (c) SIMD; (d) MIMD.

2.2.3 Händler's Taxonomy The taxonomy of Wolfgang Händler [Hän77],presents another method for determining the classification of a computer system. Händler's taxonomy is an attempt to classify a system by its degree of parallelism and its pipelining capabilities. *Pipelining* is defined as the system's ability to decompose a process into distinct subprocesses which may be executed in an overlapped manner. Under Händler's proposal, a system can be represented by a triple, $T(C)$, which contains six independent variables. $T(C)$ is defined as follows:

$$T(C) = < K \times K', D \times D', W \times W' > \tag{2.1}$$

where

K is the number of processor control units (PCU)

K' is the number of PCUs that can be pipelined together

D is the number of arithmetic logic units (ALU) per PCU

D' is the number of ALUs that can be pipelined together

W is the basic wordlength of the ALU

W' is the number of pipeline stages in the ALU

The significance of Händler's taxonomy is that it introduces the concept of pipelining as a classification measure.

2.3 Interconnection Networks

In a multiprocessor environment, the ability of a particular processor to communicate with other processors in the system is dependent upon the topology of the network which connects them and the interprocessor communication switching methodology. Interconnection networks can range from simple and inexpensive to complex and cost prohibitive. The most simple (logically) interconnection network is the ring. The complexity of a ring is proportional to the number of processors in the ring, O(n). As its name implies, the ring interconnection network forms a closed looped by connecting neighboring processors in a uni- or bi-directional ring. The communication time in the ring is a function of the number of processors in the ring. As a result, systems implemented using the ring interconnection topology are severely limited in the number of processors which can be connected. Therefore, the ring interconnection can be feasibly applied only when the number of processors to be connected is small. On the other end of the complexity-cost spectrum lies the crossbar switch. The crossbar switch is characterized by n inputs and n outputs. An n-by-n crossbar switch allows for full-connectivity between its n inputs and n outputs. As a result of having the capability of routing any input to any output, the benefits of connectivity must be paid for in logic complexity and high cost. The high cost results from a circuit complexity which is proportional to the square of the number of processors and memory devices connected to its input/output ports, O(n^2). To overcome the restrictions of small sized systems inherently related with ring interconnection networks, and cost prohibitive systems implemented solely with crossbar switches, design compromises have to be made. As a result of these compromises, two classes of interconnection networks are being designed and constructed to allow for large numbers of processors at a reasonable cost. These two interconnection networks classes are the direct or single-stage networks, and the indirect or multistage networks. Direct networks use point-to-point links to connect the processing elements. Indirect networks, on the other hand, uses the network as a separate entity. The processing elements are connected to the inputs and the outputs of the network.

Four methods of interprocessor communications exist: *circuit switching*, *packet switching*, *virtual cut through*, and *wormhole routing*. The first type, circuit switching, is where a dedicated path is established prior to the transmission of data from source to destination. In circuit switching, the dedicated path is held until the transmission of data is complete. The second type of switching is packet switching. Packet switching is a concept in which messages are broken into submessages (packets) and the packets, along with their routing information are allowed to independently traverse across the network from input to output. In virtual cut through, the packet headers are examined to determine the next appropriate channel for the packet to be transmitted on. Virtual cut through uses a store and forward method of transmitting the packets once the packet header has been examined. If a blockage exists, the packet is buffered until the blockage has been resolved. Wormhole routing uses the same basic approach of examining the packet header as virtual cut through. The two methods differ in how the packet is handled when a blockage is encountered. Instead of buffering the packet as done by virtual cut through, wormhole routing keeps the packet in the network until the blockage is resolved.

2.3.1 Single-Stage Networks The single-stage network is considered to be a *dynamic* network with a collection of n input selectors and n output selectors [HwB84]. A dynamic network can be described as a network which has the ability to reconfigure its interconnection links. The manner in which these links are reconfigured is dependent upon the implementation of the interconnection function. Examples of single-stage networks are the Illiac IV [BaB68], the Shuffle-Exchange [Sto71], the PM2I [Sie85], and the Cube [Sie85]. Subsubsection 2.3.1.1 discusses the Illiac IV interconnection function and physical layout. The Cube network is examined in the same light in Subsubsection 2.3.1.2.

2.3.1.1 The Illiac IV Interconnection Network. The Illiac IV network received its name from the SIMD machine, the Illiac IV, designed in the late 1960s and early 1970s [BaB68]. The Illiac IV network has a physical layout which is approximately equivalent to a two-dimensional mesh. The Illiac IV network differs from a mesh network in that the border processing elements are connected in a "wrap-around" fashion. Figure 2.2 shows the physical layout of a Illiac IV network where the number of processing elements is equal to 16.

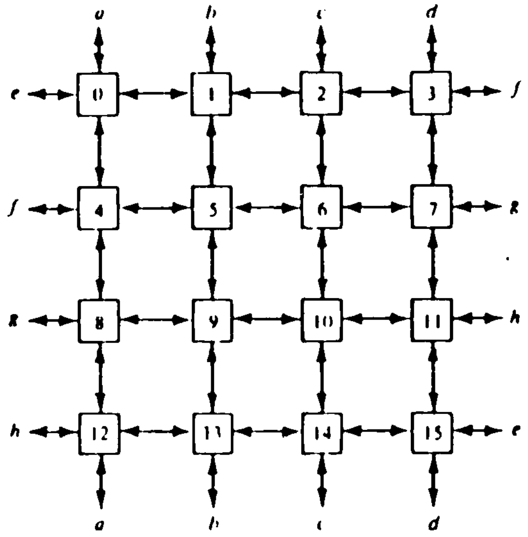


Figure 2.2. The Illiac IV network, N = 16 processors.

The physical interconnection of the Illiac IV processing elements is based on four interconnection functions. These functions are as follows:

$$Illiac_{+1}(P) = (P + 1) mod N \qquad (2.2)$$

$$Illiac_{-1}(P) = (P - 1) mod N \qquad (2.3)$$

$$Illiac_{+n}(P) = (P + n) mod N \qquad (2.4)$$

$$Illiac_{-n}(P) = (P + n) mod N \qquad (2.5)$$

where

P is the processor identification number

N is the number of processors in the Illiac IV network

n is the $\sqrt{N}$

mod is modulus arithmetic

2.3.1.2 The Cube Interconnection Network The Cube network [Sie77] is a single-stage interconnection network whose name is derived from its processing elements' physical interconnection pattern. The *dimension* of the cube is determined by the number of processing elements in the cube. As an example, let N be the number of processing elements in the cube. The dimension of the cube, m, is $log_2(N)$. In an m-dimensional cube, the processing elements are located on the vertices of the cube. Each processing element is connected to m adjacent processing elements. A 3-dimensional cube is shown in Figure 2.3. The interconnection of the processing elements in an m dimension cube can be described by m interconnection functions.

$$Cube_k(p_{m-1}, p_{m-2}, \ldots p_1, p_0) = p_{m-1}, p_{m-2}, \ldots \overline{p_k}, \ldots p_1, p_0 \,, \; 0 \leq k < m \qquad (2.6)$$

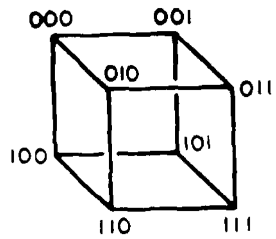


Figure 2.3. The Cube interconnection network where m = 3, N = 8.

where p_i is the i^{th} bit of a processing element's address. The $Cube_k$ function connects a particular processing element, represented on the left side of the equation, to a processing element given on the right side of the equation. The two processing elements' addresses differ in the k^{th} bit position.

2.3.2 Multistage Networks

Like the single-stage networks, multistage networks are dynamic networks. Multistage networks can be described by three characterizing features: the *switching element*, the *network topology*, and the *control structure* [HwB84]. The switching element is a device whose function is to interchange its p inputs and p outputs.

As an example, a 2-by-2 switch box has four allowable settings: *straight, exchange, upper broadcast,* and *lower broadcast,* as shown in Figure 2.4.

Multistage networks can be one- or two-sided. One-sided networks are those whose input and output ports are on the same side of the network. Two-sided networks have inputs on one side of the network and outputs on the other side. Two-sided multistage networks can be categorized into three classes: *blocking, rearrangeable,* and *nonblocking* [Fen81].

Blocking networks are those in which the connection of more than one terminal pair simultaneously, may cause conflicts in the allocation of the remaining communication links. Examples of blocking networks are the Data Manipulator [Fen81], the Omega [Law75], and Indirect Binary n-cube [Pea77].

Rearrangeable networks are those which can perform all possible combinations of connections between inputs and outputs by rearranging existing connections to allow for new input-output connections. The Benes network [Ben65] is an example of a rearrangeable network.

The third class of two-sided multistage networks is the nonblocking network. In a nonblocking network, there exists a one-to-one connection between input and output port. The crossbar switch which provides full-connectivity between inputs and outputs is an example of nonblocking network. A multistage network with N processing elements will contain at least $log_p N$ stages, where p is the size of the crossbar switching box. Each stage of the network will consist of N/p switching boxes.

A third feature used in characterizing multistage networks (the switching elements and the network topology being the first two) is the control structure. The control structure of the network determines how the switching elements are to be controlled. There exists two basic methods for implementing the control structure: *distributed* or *centralized* control. In a distributed control structure, each switching

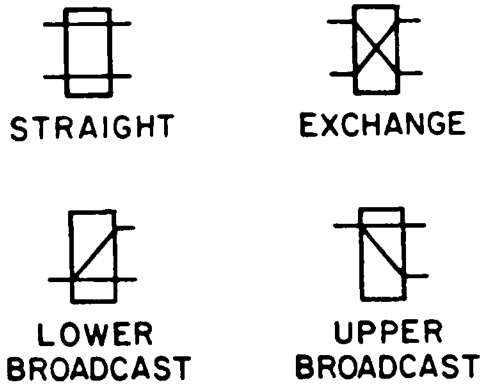


Figure 2.4. The four settings of a 2-by-2 switch box

box contains control logic which uses routing information contained in the header of a message to determine the setting of the switching box. Centralized control uses a centrally located control unit to inform individual switching boxes of their routing settings. Implementation tradeoffs must be considered when choosing the control structure for a multistage network. While the advantages of constant path set-up and simple interchange box logic make centralized control seem more preferable than distributed control, centralized control disadvantages far outweigh its advantages over distributed control. The major disadvantage of centralized control is that only one message can be routed at any instance of time, thereby serializing the network accesses. Using distributed control, multiple messages can be routed simultaneously yielding no bottleneck effects.

2.3.1.3 The Multistage Cube Interconnection Network. The multistage cube network [McS81, Sie85] is based on the Cube interconnection function presented in Section 2.3.1.2. Its topology is equivalent to the blocking networks characterized above. The multistage cube network consists of log_pN stages, where N is the number of processing elements in the system and p is the size of the crossbar switching element. Each stage in the network contains N/p switching elements. Each stage of the multistage cube implements the Cube function. By this, the boxes of the i^{th} stage implements the $Cube_i$ function. At stage i, the address lines that differ in the i^{th} bit position are paired at the switching elements. Figure 2.5 shows the multistage cube network for $N = 8$ implemented with 2-by-2 crossbar switching elements.

2.4 Parallel Processing Systems

Contemporary parallel processing systems have been implemented to take advantage of the architectural advances made in the design of interconnection networks. This section reviews five major systems which have either been implemented commercially or have been built solely for the purpose of research of the present technology.

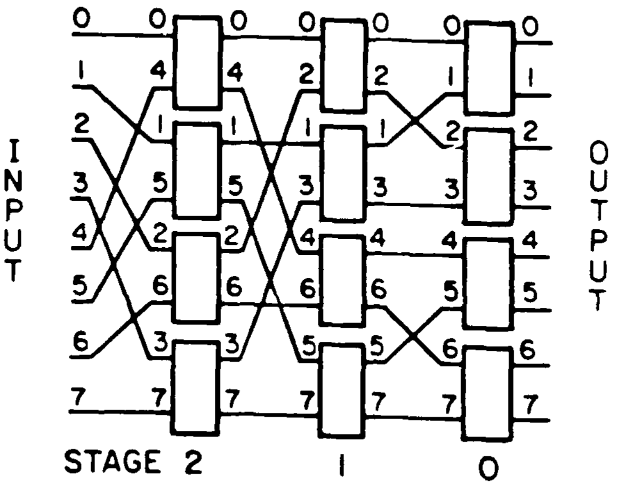


Figure 2.5. The multistage cube network, N = 8.

From a network point of view, parallel processing systems can be grouped into one of two architectural categories: processor-to-memory (P-M) or processing element-to-processing element (PE-PE, where a PE is a processor-memory pair). Processor-to-memory architectures use bi-directional networks to connect processors to memory modules. Processor-to-memory architectures are characterized by heavy network loading which results from inter-processor communications and memory accesses across the network. In a PE-to-PE architecture, the network is unidirectional and provides inter-PE communications only. The PE-to-PE architecture differs from the P-M architecture in that no commonly accessible memory modules exist. As a result, the network loading is less in a PE-to-PE system than in a P-M system. Figures 2.6 and 2.7 show the PE-to-PE and P-M architectures respectively.

2.4.1 The Illiac-IV The Illiac-IV, a SIMD machine, was developed in as joint effort between the University of Illinois and the Burroughs Corporation [Sto77]. Proposed in 1965 and shipped in 1972, the Illiac-IV was one of the first machines to be implemented using a parallel architecture. Original proposals were for the Illiac-IV to be a multi-SIMD machine with four quadrants, each of which would contain 64 processing elements. Only one quadrant was ever constructed. The Illiac-IV was primarily designed to solve partial differential equations and perform matrix multiplication. The interconnection network of the Illiac-IV was a variation of the mesh interconnection network.

2.4.2 The BBN Butterfly The BBN Butterfly is another parallel machine whose interconnection network implementation is the multistage cube. Manufactured by Bolt, Beranek, and Newman, Inc., the Butterfly is designed for commercial time-sharing use as well as for image processing in a research environment [CrG85].

The Butterfly is designed to house up to 256 independent processors. At present, the machine has been commercially packaged to contain from 1 to 128 processors. As mentioned above, the interconnection network is a multistage imple-

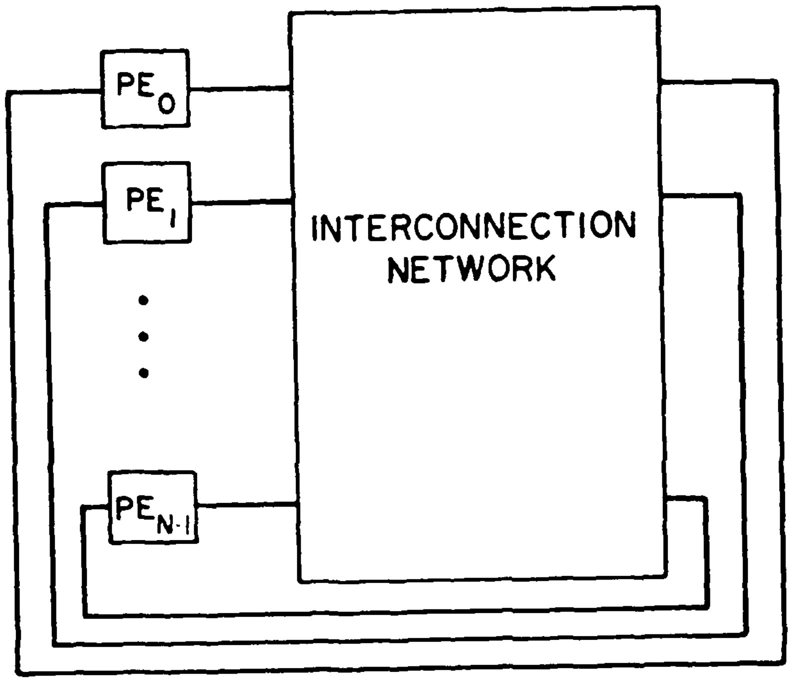


Figure 2.6. PE-to-PE system architecture.

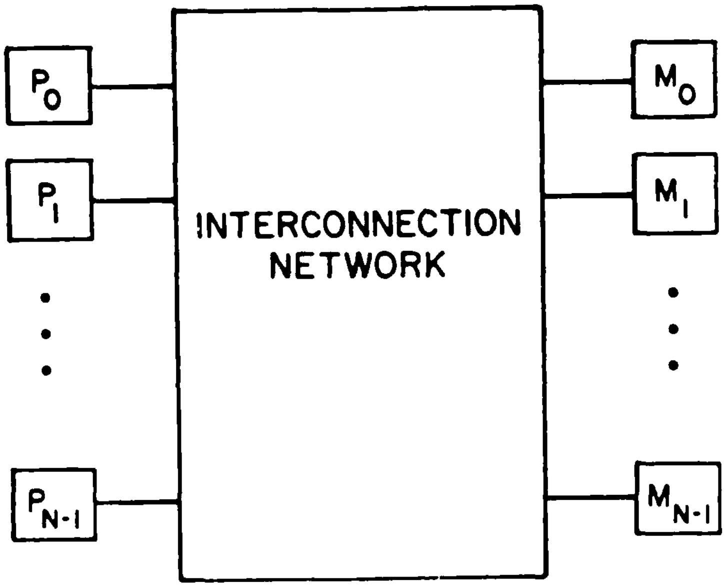


Figure 2.7. Processor-to-Memory system architecture

mentation. The interchange boxes are 4-by-4 crossbar switches. Packet switching and distributed routing control are used for message transmission. The Butterfly's system architecture is a PE-to-PE architecture.

2.4.3 The NYU Ultracomputer The Ultracomputer is a shared-memory MIMD machine which is presently under development at New York University [GoG83]. This machine, when fully implemented, will house 4096 autonomous processors for use as a general purpose parallel system. At the present, an Ultracomputer prototype containing 64 processors has been built. The Ultracomputer uses a P-M systems architecture.

The interconnection network of the Ultracomputer is a multistage cube which uses 4-by-4 crossbar switches as its switching elements. The switching methodology used by the Ultracomputer is packet switching with the routing of the packets using the destination address routing method to traverse the network.

2.4.4 The Intel iPSC The Intel Personal Super Computer (iPSC) is a research-oriented MIMD machine. The iPSC architecture is more commonly referred to as the *hypercube* or *binary n-cube* based upon its interconnection network, a packet switched implementation of a single-stage network. The hypercube may consist of one, two, or four 32-node computational units. Each of the cube's 32 processing nodes can function independently and concurrently with one another. A central controller (cube manager) serially passes data and processes code to active nodes within the cube.

2.4.5 The IBM Research Parallel Processor Prototype (RP3) The RP3 is a MIMD machine designed strictly to research the hardware and software aspects of parallel processing [PfB85]. Incorporating much of the NYU Ultracomputer design, the RP3 has been designed and is being constructed at the IBM T.J. Watson Research Center. The RP3 will contain 512 32-bit microprocessors. These 512 processors will be grouped into eight modules with each module containing 64 processors.

The interconnection network of the RP3 consists of two separate networks: a multistage cube and a combining network which is used for interprocessor coordination functions. As with the Ultracomputer and the Butterfly, the RP3's interchange boxes are implemented using 4-by-4 crossbar switches. The interprocessor communications are a mixture of circuit- and packet-switching.

2.5 Summary

In this chapter, an overview of parallel processing systems has been presented. Three methods for classifying parallel processing systems were discussed. While Flynn's taxonomy is the most widely recognized of the three methodologies, it fails to provide architectural details of a system. Both Feng and Händler provide limited architectural insight, but fail in providing enough information to accurately describe a system.

Three interconnection networks were discussed from a functional implementation point of view. Interconnection network communication switching methodologies were also defined. Each network's interconnection function was presented along with its physical layout. Multistage networks were discussed along with a presentation of their associated interchange boxes, topology and control structures.

Five parallel processing systems were briefly examined from a networks point of view. These systems revealed the progression of parallel systems architectural implementations. This progression began with the first implemented parallel machine, the Illiac IV, and has proceeded to present-day systems such as the NYU Ultracomputer, INTEL's iPSC, and IBM's RP3.

3. Performance Modeling and Analysis

3.1 Introduction

Over the past six years, extensive research has examined interconnection networks from a performance modeling and analysis viewpoint. These performance studies have ranged from analytically modeling the probability of message collisions in crossbar switches, to determining which of the two switching methodologies is best suited for interprocessor communications. Further studies have examined the possibilities of modeling interconnection networks via computer simulations. Interconnection network comparison studies have proven valuable in assisting system architects choose the network which will best suit the application. The following sections review previous research performed in performance modeling and analysis.

3.2 Crossbar Switch Analysis

The crossbar switch, defined and discussed in Chapters 1 and 2, is used as the basic switching element in many multiprocessor designs and implementations [AdS84, CrG85, Dav85, DaS85, DaS86, DiJ81, GoG83, Law75, McA81, Pea77, PfB85, Sie85, SiH86]. This section presents a review of the research performed by Patel [Pat81]. Using analytic and simulation techniques, Patel compares the Delta networks, a permutation of the multistage cube network [Sie85], and networks comprised solely of crossbar switches. This review of Patel's work is concerned only with the implementation issues and analytical analysis associated with the use of a crossbar switch as a network switching element. For consistency, the notation of Patel is used in the following discussion.

From a hardware point of view, the crossbar switch consists of two major components: the control logic and the switching element itself. The control logic is used to process message requests and to provide arbitration among requestors in the event of a conflict. The function of the switching element is to route the message or data once the setting of the switch has been determined by the control logic. A functional block diagram of a 2-by-2 crossbar switch is shown in Figure 3.1.

The single lines in Figure 3.1 represent one bit lines. The double lines into and out of the INFO box represent address lines, data lines, and a Read/Write control line. The X and $\overline{X}$ lines are used to control the switch setting. If the input X is logic 1, the switch is set to its cross connection. If X is logic 0, the switch is set to its straight connection (see Figure 2.4). For the 2-by-2 case, only one bit has to be examined to determine the setting of the switch. Switches of size N require $N - 1$ bits to be examined to properly set the crossbar connections. In the implementation of a crossbar switch, two sets of control lines exist: the request, the destination, and the busy lines for the left or input side of the switch and the request and the busy lines for the right or output side of the switch. For the 2-by-2 sized crossbar, two sets of these control lines exist, one set for each of the input and a set for the output lines of the switch. An N-by-N crossbar switch requires N sets of control lines. The logic equations for the signals in Figure 3.1 are given below.

$$\begin{aligned}
X &= r_0 d_0 + \overline{r_0 d_1} \qquad (3.1)\\
\overline{X} &= r_0 \overline{d_0} + \overline{r_0} d_1\\
R_0 &= r_0 \overline{d_0} + r_1 \overline{d_1}\\
R_1 &= r_0 d_0 + r_1 d_1\\
b_0 &= \overline{X} B_0 + X B_1\\
b_1 &= X B_0 + \overline{X} B_1 + r_0 d_0 d_1 + r_0 \overline{d_0 d_1}\\
I_0 &= i_0 \overline{X} + i_1 X\\
I_1 &= i_0 X + i_1 \overline{X}
\end{aligned}$$

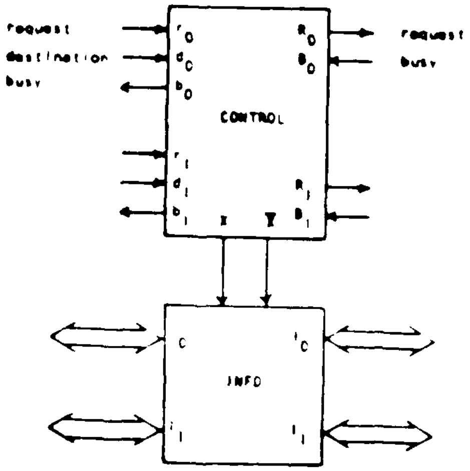


Figure 3.1. 2-by-2 crossbar module [Pat81].

The above equations represent the logic equations for the simplest crossbar, the 2-by-2. For large N, the logic equations for an N-by-N crossbar become complex to the point of intractable.

In the analysis of the crossbar, a crossbar size of M-by-N is assumed. Patel's analysis assumes a processor to memory system architecture (see Figure 2.7). Using this assumption, the crossbar supports M processors and N memory modules. Conflicts caused by two requests made for the same memory module are to be considered memory conflicts rather than network conflicts. Further assumptions are made to facilitate the analysis. First, each processor generated requests randomly and independent of the others with the requests uniformly distributed over the memory modules. A second operating assumption is that in each cycle, each processor generates new requests with a probability m. Using these assumptions, the bandwidth (BW), in packets per unit time, of the crossbar and the probability of message acceptance (P_A), at the destination memory module, can be derived for crossbar switches of varying size. For large M and N, the bandwidth and probability of message acceptance is represented by:

$$BW \simeq N(1 - e^{-mM/N}) \qquad (3.2)$$

$$P_A \simeq \frac{N}{mM}(1 - e^{-mM/N}) \qquad (3.3)$$

The complete derivation of the above equations can be found in [Pat81]. The above approximations provide 99% accuracy when M and N are greater than 30 and 95% accuracy for $M, N \geq 8$.

3.3 *Circuit Switching Analysis*

Of the four types of switching methodologies presented in Chapters 1 and 2, two have been predominantly used in interconnection network analysis: circuit switching and packet switching. In this section and the one to follow, a review and analysis of previous research in these two areas is presented.

In a circuit switched network implementation, a physical link between source and destination PEs is established prior to message transmission. The path establishment process is performed using a "request-grant" protocol [McS80]. In a distributed control system, message requests are forwarded through the network, setting the switching elements to the appropriate position if available. Once the transmission path is established, it is held until the transmission of the message(s) is complete. In cases where two or more different messages desire the same communication path or a blockage occurs due to a previously established path, a conflict among the message transmissions arises. In the event of a conflict, a method for resolving the conflict must be chosen.

Two of the more common conflict resolution methods, the drop and the hold algorithms, have been the topics of recent research [Dav85, LeW84]. When a conflict is encountered using the drop algorithm, the message request is removed from the network and the partial communication link, up to the point of the conflict inclusive, is relinquished. Message requests that are dropped from the network must be reinitiated by the originator at a later time. The hold algorithm differs from the drop algorithm in that when a conflict arises, the message request is held at the point of conflict until the conflict has been resolved. The partial communication link is held intact.

In the work of [Che82, ChL83, Dav85, LeW84], each used discrete time Markov chains to aid in the analysis of the circuit switched networks implementing either the drop or the hold resolution algorithms. Modeling a system using Markov chains can provide a graphical representation of the operating states of the network as well as the state transition probabilities.

3.3.1 The Hold Conflict Resolution Algorithm This section presents the results of the mathematical derivations performed by [Dav85, LeW84] in determining the state transition equations for the hold conflict resolution algorithm. For consistency in the presentation of these results, the notation of [Dav85] is used.

Using the Markov chain shown in Figure 3.2, the hold conflict resolution algorithm can be modeled for a 4-stage network. The derivations of the state transition probabilities assume a 2-by-2 size switching element is implemented in the network. The network can be modeled by using two sets or states: the request states, R_i, and the blocked states, B_i, $0 \leq i < n$. These states represent the possible states that a message request may encounter as it attempts to traverse the network and establish a path. State P represents the processing state of the message at a local PE. State T represents the state in which the data transfer may begin. This state signifies the establishment of the communications link between source and destination. Assuming a message generation rate, m, from each source PE, and a time delay, d, associated with the state T, the state transition probabilities are given below in Equation 3.5.

$$\begin{aligned} q(T,P) &= 1/d \qquad (3.4) \\ q(T,T) &= 1 - 1/d \\ q(P,R_0) &= m \\ q(P,P) &= 1 - m \end{aligned}$$

In deriving the transition probability of moving from state R_i to state R_{i+1}, the three possible causes for blockage in a 2-by-2 switching element were summed. The resulting probability of transitioning from one stage to the next is given by:

$$q(R_i, R_{i+1}) = 1 - .25R_i - B_i - .5\sum_{j=i+1}^{n-1}(R_j + B_j) - 0.5p(T) \qquad (3.5)$$

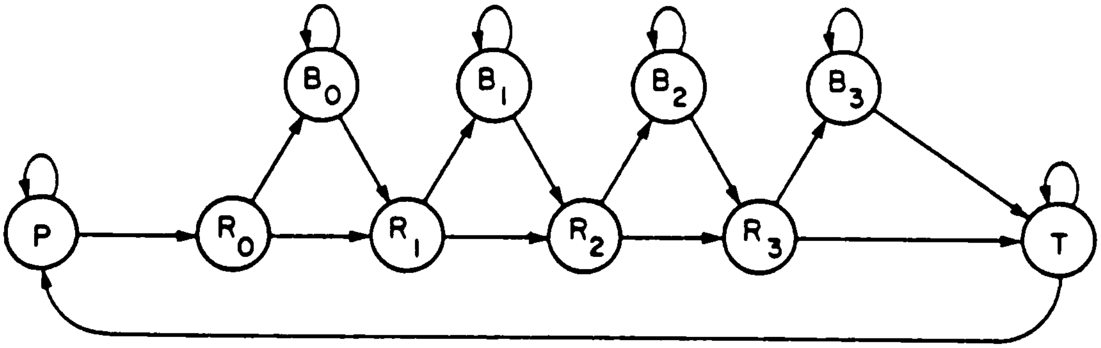


Figure 3.2. Markov chain for a 4-stage (N=16) network using the hold conflict resolution algorithm [Dav85].

The transitions from state R_i to state B_i was defined as follows:

$$q(R_i, B_i^j) = \begin{cases} .25p(R_i) - p(B_j) & j = i \\ .5(p(R_j + p(B_j)) & i < j < n \\ .5p(T) & j = n \end{cases} \tag{3.6}$$

In Equation 3.6, the superscript notation, B_i^j, is used to indicate that the blocked message is in state j. The superscript changes as the blocked message moves from state to state.

The probability of transitioning from one stage to the next stage in a particular time cycle t was defined to be:

$$q_{jt} = \frac{p_{t+1}(R_{j+1})}{p_t(R_j) + p_t(B_j)} \tag{3.7}$$

Equation 3.7 shows that the probability of transitioning through stage j is the probability of being in stage j divided by the probability of being in state R_{j+1}. Using this time-based probability, q_{jt}, the transition probabilities for state B_i^j were shown to be:

$$q(B_i^j, R_{i+1}) = \begin{cases} 1/d & j = n \\ 0 & j < n \end{cases} \tag{3.8}$$

$$q(B_i^j, B_i^{j+1}) = \begin{cases} q_j & i < j \leq n \\ 1 - q_{j+1} & j = i \end{cases} \tag{3.9}$$

$$q(B_i^j, B_i^{j+2}) = \begin{cases} 0 & i < j \leq n \\ q_{j+1} & j = i \end{cases} \tag{3.10}$$

3.3.2 The Drop Conflict Resolution Algorithm Using an approach similar to their analysis of the hold conflict resolution algorithm, Lee and Wu [LeWS4] present a two dimensional structure which models the message request transitions through the network when the drop conflict resolution algorithm is used. The Markov chain representation for a 4-stage network is shown in Figure 3.3. As pointed out in [Dav85], this representation is an approximation to the actual system operation in that multiple blocking messages are not addressed.

The states P and T of Figure 3.3 represent the same states as those in the hold algorithm analysis: the local processing at the PE and the message transmission. The states $R_{i,j}$ represent the traversal of the message requests through the network. The $R_{i,j}$ states encompass the record keeping of the location of a blocked message requests, which in expanded notation is represented by a third subscript, k. The first row of the model represents the first transmission attempt of a message request entering the network. Subsequent rows depict the retransmission of the message request following blockage in the previous row. Column states are used to model the message requests at a particular stage of the network.

As with the hold algorithm, the transition probabilities between the message processing state, P, the transmission state, T, and the first path request state are the same:

$$\begin{aligned} q(T,P) &= 1/d \\ q(T,T) &= 1-1/d \\ q(P,R_{0,0}) &= m \\ q(P,P) &= 1-m \end{aligned} \tag{3.11}$$

Letting T be denoted by stage n, the probability of being in column j is defined as:

$$S_j = \sum_{i=0}^{n} p(R_{i,j}) \tag{3.12}$$

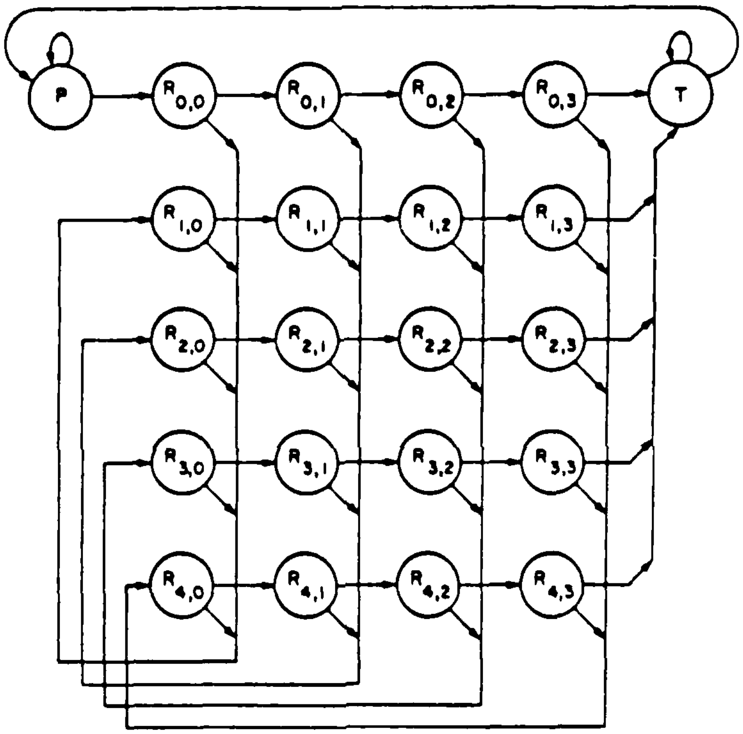


Figure 3.3. Markov chain representation of a 4-stage ($N=16$) network using the drop conflict resolution algorithm with dependent message resubmissions [Dav85].

Similar to the derivations in the hold algorithm, the transition probabilities through the first row are found to be:

$$q(R_{0,j}, R_{0,j+1}) = 1 - .25p(S_j) - .5 \sum_{m=j+1}^{n-1} S_m - .5p(T) \tag{3.13}$$

When a blockage occurs, the message request is dropped from the network and resubmitted in the row that corresponds to the column where the blockage occurred. The transition probabilities are represented by:

$$q(R_{i,j}, R_{j+1,0,k}) = \begin{cases} .25S_j & k = j \\ .5S_k & j < k < n \\ .5p(T) & k = n \end{cases} \tag{3.14}$$

Equations 3.13 and 3.14 represent all possible transitions in the network for message requests that are independent of other requests that may be in the network. Transitions that occur when a previously blocked message request returns to the column of its initial blockage are considered to be dependent transitions [Dav85]. To facilitate the analysis of these dependent transitions, the probability of transitioning through a column j in one time cycle is defined as:

$$q_{jt} = \frac{S_{(j+1)t+1}}{S_{jt}} \tag{3.15}$$

From Equation 3.15, q_j is the probability that the message request did not complete its transmission in the present time cycle, meaning that the message is still active in the system. The probabilities q_j are shown to be:

$$q_j = 1 - 1/d \qquad n \leq j \tag{3.16}$$

Using the representation for q_j, the probability of the blocking request being active in the network after the present time cycle is:

$$Q_{k,j} = \prod_{m=k}^{k+j} q_m \tag{3.17}$$

$Q_{k,j}$ can be described as the probability of the blocking message transitioning from column k to column k+j while the blocked message is resubmitted and returned to the point of blockage [Dav85]. Using these derivations, the transitions for the dependent states, $R_{j+1,j}$, can be derived. First, Davis derives the transition probabilities of not being blocked and transitioning through the column. These probabilities are found to be:

$$q(R_{j+1,j,k}, R_{j+1,j+1}) = \begin{cases} (1 - Q_{k,j})(1 - .25S_j - .5\sum_{m=j+1}^{min(2j,n)} S_m) & k \neq j \\ (1 - Q_{k+1,j-1})(1 - .25S_j - .5\sum_{m=j+1}^{min(2j,n)} S_m) & k = j \end{cases} \tag{3.18}$$

The transitions resulting from the repeated blockage due to the blocking message and the blockage by a new request are given in Equations 3.19 and 3.20.

$$q(R_{j+1,j,k}, R_{j+1,0,m}) = \begin{cases} Q_{k,j} & m = min(k+j, n), \quad k \neq j \\ Q_{k+1,j-1} & m = min(k+j, n), \quad k = j \end{cases} \tag{3.19}$$

$$q(R_{j+1,j,k}, R_{j+1,0,m}) = \begin{cases} .25(1 - Q_{k,j})S_j & m = j, & k \neq j \\ .5(1 - Q_{k,j})S_m & j < m \leq n, & k \neq j \\ .25(1 - Q_{k+1,j-1})S_j & m = j, & k = j \\ .5(1 - Q_{k+1,j-1})S_m & j < m \leq n, & k = j \end{cases} \tag{3.20}$$

And finally, blockage transitions in the input stage were derived in [LeW84] and are given below.

$$q(R_{1,0,n}, R_{1,0,n}) = 1 - q_n \tag{3.21}$$

$$q(R_{1,0,k}, R_{1,0,k+1}) = q_k \tag{3.22}$$

$$q(R_{1,0,k}, R_{1,0,0}) = .5(1 - q_k) \tag{3.23}$$

$$q(R_{1,0,k}, R_{1,1}) = .5(1 - q_k) \tag{3.24}$$

$$q(R_{1,0,0}, R_{1,0,1}) = 1 \tag{3.25}$$

3.4 Packet Switching Analysis

Packet switched network performance and analysis has been the topic of extensive research in recent years [DiJ81a, DiJ81b, MuM82, Che82, KrS83, ChH84, DaS86]. In this section, the basic principles of packet switched networks are examined along with the techniques used to model this type of network implementation. Results from previous research will also be discussed.

Packet switched networks differ from circuit switched networks in the manner in which the communication link between source and destination pairs are maintained. While the complete link is held until a message has completed its traversal of the network in a circuit switched environment, the link between switching elements is held just long enough for the message to traverse the link in a packet switched network. This eliminates the need for a complete path from source to destination prior to the message transmission.

Advantages and disadvantages exist when using packet switched networks instead of circuit switched implementations. An advantage of packet switched networks is the ability to pipeline the packet transmissions thereby potentially reducing the overall transmission delays and increasing the network throughput. With these benefits come potential drawbacks of this implementation. Since each packet establishes its own path through the network, the path establishment delays increase as the number of packets in a message grows. Additionally, the logic required to control the switching elements is more complex than a comparable network constructed using the circuit switched methodology.

As in the circuit switched network, conflict resolution algorithms exist in packet switched networks. When a conflict occurs at a switching element, one of three resolution algorithms can be used: the hold, the drop, and the reroute algorithms. The hold and the drop algorithms are the same as those discussed in the previous section. When the reroute algorithm is used, a blocked message is reroute to an incorrect destination for resubmission by that destination to the correct destination.

Methods for modeling packet switched networks which have used both discrete and continuous time Markov chains have been presented in [DiJ81a, DiJ81b, Che82, KrS83, ChH84, Dav85]. Queueing models consisting of n nodes which represent the n stages of the network can be used to analyze the packet traversal of the network. Switching elements are represented by queues at the inputs of the n nodes. Queue lengths are assumed to be of a finite length determined prior to implementation.

Dias and Jump [DiJ81a, DiJ81b] investigated the affects on the network due to varying buffer sizes. By doubling the buffer size from one to two packets, they were able to show that the throughput of the network is also increased. The increase in throughput continues, to a point, as the size of the buffers is increased. The throughput begins to remain approximately constant with buffer sizes in excess of 4-6 packets. It is also shown that packet delays increase as the buffer sizes increase. The bottom-line of the research of Dias and Jump, shows that an optimal buffer size exists to maximize the network throughput. Additional research performed on multiple-packet networks, by [DaS86], is presented in the sections which follow.

3.5 Tradeoff Analysis of Switching Methodologies

In this section, a tradeoff analysis of circuit-switched versus packet-switched multistage generalized cube network is reviewed. Two operating modes are presented in the original analysis: the SIMD mode and the MIMD mode. This review is only concerned with the MIMD mode of operation due to its relationship to this thesis investigation.

Davis and Siegel [DaS86] perform a comparative study into the effects of choosing packet-switching versus circuit-switching as the switching methodology for the multistage cube network. The effects of multiple-packet messages on the network are also researched. Results from this research are detailed in the text which follows.

In a MIMD environment, the generation and transmission of messages to and through the network occur asynchronously. Messages generated consist of a header, containing the routing information, and one or more data words. If the size of the message exceeds the maximum single-packet size allowed by the network, the message must be broken-down into multiple packets for transmission through the network. Each packet contains the same routing information. In the multistage cube network, a single path exists between a source and destination pair [McS81]. When multiple packets occur, these packets must be routed sequentially to ensure proper ordering at the destination.

Davis and Siegel, in analyzing the performance of multiple-packet messages, introduce two parameters to aid the analysis. The *packet cycle time* is defined as the time delay associated with the packet moving from an input to an output of a network interchange box. The *packet offset time* is considered to be the time between successive packet generations in a multiple-packet message. This time can be further described as the time difference in the speed of the system PEs and the network. As an example, the packet offset time is equal to one when the time to generate a packet is equivalent to the time required by an interchange box to process the packet.

Results from their study shows that, for a given message size, the delay of a packet in the network decreases as the packet offset increases. This is to be expected, as an increase in the packet offset reduces the apparent network loading and subsequently, the network conflicts. These values can be used to compare against the ideal times required by a packet to traverse the network. Under ideal conditions, an m-packet message requires k+(m-1) packet cycles to traverse the k-stage network.

When making the choice of packet-switching or circuit-switching for network implementation, different factors must be considered. First, the operational mode (SIMD or MIMD) will determine the effects on the network due to conflicts among the messages and the associated queueing delays. A second issue is the type of systems architecture that supports the network. The two types, PE-to-PE and P-M system architectures, discussed in Chapter 2, determine whether the network will be characterized by light loading and low conflicts (PE-to-PE), or more heavily loaded with greater conflicts in a P-M architecture. Through cacheing techniques, networks supporting P-M systems are shown to perform equivalently to those in PE-to-PE architectures.

To compare a circuit-switched network to a similar network implemented using packet-switching, the internal and external environment must be the same in both cases. Design implementations such as data path width, interchange box implementations, and PE-network interfacing techniques are internal factors that must be considered. External factors that must be the same for comparative purposes are: system size, processing speeds and network loading. Once these factors are determined and set, a valid comparison of the switching methodologies can be made.

Davis and Siegel conclude that the circuit switched network provides better performance for smaller-sized messages than do the packet-switched network. The packet-switched network performs better for messages of longer lengths. These results also show that the performance of the network is highly influenced by the processing rates of the PEs.

3.6 Comparisons of Interconnection Networks

The ability to accurately compare interconnection networks which differ topologically is essential in determining the suitability of the network to a particular application. For this reason, recent research efforts have been directed toward these types of comparisons. This section reviews the work of Dally [Dal86], Abraham and Padmanabhan [AbP86], and Hsu, Yew, and Zhu [HsY87]. Two of these studies [Dal86] [Szy86], perform comparisons of networks based on VLSI design constraints. Analytical modeling and analysis of network performances are examined by [AbP86]. Comparisons of the single stage cube (hypercube) network to a newly proposed network is the topic of research of [HsY87].

3.6.1 VLSI Comparison of Tori and Binary n-Cube Networks W. J. Dally [Dal86] presents a comparison of interconnection networks based on the wiring requirements of VLSI circuits used to implement the networks. In his study, Dally compares low-dimensional networks (e.g., mesh) to high-dimensional networks (e.g., binary n-cubes), with each having the same *bisectional width*. The bisectional width of a network [Tho80] is the minimum number of wires that must be cut if the network is to be divided into two equal halves. This comparison is further based of three performance parameters: *latency*, *average case throughput*, and *hot-spot throughput*. Latency is the time interval between successive initiations. The average case throughput is defined as the average number of messages processed by the network in a unit of time. A measure of the throughput between a pair of processing elements which receive a disproportionately large amount of the network traffic is called the hot-spot throughput.

One of the main operating assumptions used in Dally's study is the use of wormhole routing. Wormhole routing, recall from Chapter 2, is a variation of virtual cut through routing [KeK79]. These two methods differ in the manner in which a blocked message handled. While virtual cut through removes a blocked message from the network, wormhole routing retains a blocked message in the network. The benefits of using wormhole routing over the store-and-forward routing method is reduced network latency. This is shown mathematically below in Equations 3.26 and 3.27 and graphically in Figure 3.4.

The derivation of the network latency using the store-and-forward and wormhole routing techniques is dependent upon two components of latency: the distance (D) and the message aspect ratio (L/W). The distance, D, is defined as the mean point-to-point distance (in hops) from source to destination. The message aspect ratio is the ratio of the message length, L, over the normalized channel width, W, and can be described as the number of channel cycles necessary to transmit the message across one channel. Using the store-and-forward method, an entire message must be received by an intermediate node prior to the message being transmitted to the next node in the communication link. The latency of the network must then be the product of the distance through the network and the message aspect ratio.

$$T_{store-and-forward} = T_{channel}(D \times \frac{L}{W}) \tag{3.26}$$

Using wormhole routing, partial messages may be transmitted upon receipt of the control bits (flits) to the next node in the communication link. The latency of the network can now be represented by the sum of D and L/W.

$$T_{wormhole} = T_{channel}(D + \frac{L}{W}) \tag{3.27}$$

In Equations 3.26 and 3.27, $T_{channel}$ is the channel cycle time, the time needed to complete a transaction on a channel. Figure 3.4 shows that the latency time required for routing a message through three processing nodes is greatly reduced when wormhole routing is used instead of store-and-forward techniques.

Noting that for two processors picked at random from a k-ary n-cube, the average number of channels that must be traversed, D, is given by

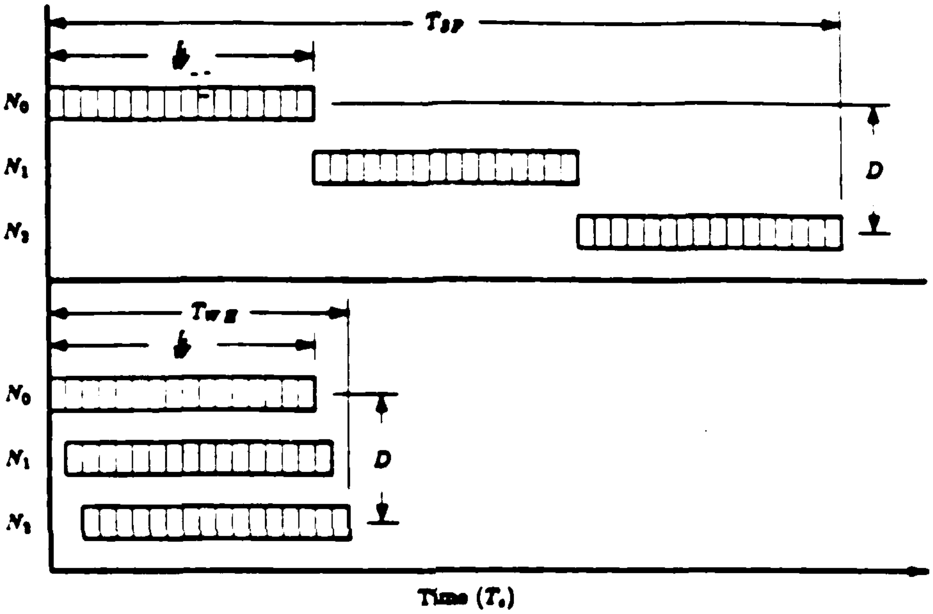


Figure 3.4. Latency of store-and-forward routing (top) vs. wormhole routing (bottom) [Dal86].

$$D = (\frac{k-1}{2})n. \tag{3.28}$$

Also the normalized channel width, W is a function of the dimensionality, n, and the radix, k of the network and can be represented by

$$W(n,k) = \frac{k}{2}. \tag{3.29}$$

Using Equations 3.28 and 3.29 substituted into Equation 3.27, the network latency for a k-ary n-cube is:

$$T_{net} = T_{channel}\left(\left(\frac{k-1}{2}\right)n + \frac{2L}{k}\right) \tag{3.30}$$

Equation 3.30 shows that the latency of the network, T_{net} is dominated by the dimension, n, of the network. This equation also assumes a constant wire length among the networks. Further derivations of the network latency include the effects due to variable wire lengths for linear and logarithmic delays associated with the lengths of the interconnection wiring. The presentation of these derivations are beyond the scope of this review. The reader should refer to [Dal86] for further information about these derivations. For each of the derivations for the network latency, the implication is that low dimensional networks will have lower latencies than the higher dimensional networks.

The second performance measure examined by Dally, is the throughput of the network. The approach taken to estimate the throughput is to calculate the capacity of the network, the maximum number of messages that can be in the network at any given instance of time.

The maximum throughput as a fraction of the capacity for k-ary n-cubes can be derived using the following assumptions and Equations 3.31 and 3.32. The results of this derivation are presented in Table 3.1.

1. Each processing node supplies to the network a traffic load of $\lambda \frac{bits}{cycle}$.
2. The message rate on channels entering the dimension is $\lambda_E = \frac{\lambda}{L} \frac{messages}{cycle}$.
3. A message traverses $\frac{k-1}{2}$ channels per dimension: one entering channel and $\sigma = \frac{k-3}{2}$ continuing channels. This gives a channel rate continuing in a dimension of $\lambda_C = \sigma \lambda_E$.
4. The service time in dimension $i+1$ is T_{i+1}.
5. The service time of the last continuing channel is the dimension is $T_{i(\sigma-1)} = T_{i+1}$.
6. The probability of collision is $\lambda_E T_{i0}$ and the expected waiting time to resolve the collision is $\frac{T_{i0}}{2}$.

Using these assumptions, the service rate for the entering channel of the dimension is given by:

$$T_{i0} = \frac{1 - \sqrt{1 - 2\lambda T_{i+1}}}{\lambda_C} \tag{3.31}$$

Equation 3.31 is only valid when $\lambda_C < \frac{T_{i+1}}{2}$. The service rate of the i^{th} channel of the dimension is given by:

$$T_i = \left(\frac{T_{i+1}}{1 - \frac{\lambda_C T_{n-1}}{2}} \right) \left(1 + \frac{\lambda_C T_{n-1}}{2} \right) \tag{3.32}$$

Setting the source service time, T_0, to the reciprocal of the message rate, λ_E and solving Equations 3.31 and 3.32 for λ_E, yields the maximum throughput of the network.

Table 3.1 shows the maximum throughput for k-ary n- cubes which support 256 and 1024 processing nodes. These calculations of total latency are for message

Parameter	256 Nodes			1024 Nodes		
Dimension	2	4	8	2	5	10
radix	16	4	2	32	4	2
Max Throughput	0.40	0.49	0.21	0.36	0.42	0.18
Latency $\lambda = 0.1$	43.9	121.	321.	45.3	128.	377.
Latency $\lambda = 0.2$	51.2	145.	648.	50.0	162.	NA
Latency $\lambda = 0.3$	64.3	180.	NA	59.0	221.	NA

Table 3.1. Maximum throughput as a function of capacity and blocking latency in cycles [Dal86].

lengths, $L = 200$ bits. Table 3.1 shows that the blocking effects due to dimensionality. are reduced as the dimension of the network is reduced.

The third performance figure examined is the hot-spot throughput. The hot-spot throughput for a k-ary n-cube which uses deterministic routing is Θ_{HS}, and equates to the bandwidth, W, of a single channel. Dally uses an assumption of constant wire cost to represent Θ_{HS} by:

$$\Theta_{HS} = W = k - 1 \tag{3.33}$$

Since low-dimensional networks have greater channel bandwidth than do high-dimensional networks, the hot-spot throughput will also be greater in the low-dimensional networks.

In this study, Dally compares low-dimensional networks (tori) to high-dimensional (k-ary n-cubes) using the assumptions outlined above. Not included in his investigation is the indirect or multistage k-ary n-cube networks. This exclusion was due to his rationale that multistage network performance is similar to the direct k-ary n-cube networks of high-dimensionality.

3.6.2 Performance of the Direct Binary n-Cube Network Abraham and Padmanabhan in [AbP86], present a mathematical analysis of the direct binary n-cube. Their investigation considers two performance measures: the probability of message acceptance and the bandwidth of the network. These performance measures, once derived, prove to be better than similar measures for the indirect binary n-cube produced by Dias and Jump [DiJ81], Patel [Pat81], and Kruskal and Snir [KrS83]. For comparison purposes, the crossbar switch sizes are limited to 8-by-8.

The analysis of the direct binary n-cube's performance is performed twice. The first analysis assumes a *single-accepting PE scheme*, where only one message can be accepted by the PE in a single cycle. The second approach assumes a *multiple-accepting PE scheme*, where up to d messages can be received by the PE in a single cycle for a d-dimensional network. Also considered are the cases where messages will and will not be buffered at the switches.

Simplifying assumptions and definitions to this research are:

1. All nodes are identical with the traffic between nodes being equally distributed.
2. The message generation rate at each PE is m_g.
3. The rate at which message arrive from neighboring node is m.
4. The rate of message entering a PE is m_a.
5. P_t is the probability that a message received from a neighboring node is for the PE.
6. P_A is the probability of message acceptance.
7. P_a is the probability that a message received from a node is successfully transmitted to another node.
8. d is the dimension of the network.
9. $N = 2^d$ is the number of nodes in the network.

	Direct Binary n-cube			Indirect Binary n-cube						
Network Size	256	1024	4096	256		1024		4096		
Switch Size	9	11	13	2	4	2	4	2	4	8
Switches	256	1k	4k	1k	256	5k	1280	24k	6k	2k
Lines	2304	11k	52k	2304	1280	11k	6k	52k	28k	20k
Crosspoints	20736	121k	676k	4k	4k	20k	20k	96k	96k	128k

Table 3.2. Hardware requirement comparisons [AbP86].

In each of the analyses, mathematical relationships are derived and used for comparison to the results obtained through simulations of the networks. Refer to [AbP86] for an in-depth presentation of the mathematical derivations. Using the above results, and through simulations, the direct binary n-cube is found to give higher P_A than the indirect binary n-cube when the size of the crossbar in the indirect network is less than an 8-by-8. Using an 8-by-8 crossbar switch reveals that the direct and indirect binary n-cubes have equal performance. The performance advantages of the direct cube for small sized crossbars do not come about without cost penalties. Table 3.2 shows various measures of costs for the two types of networks.

3.6.3 An Enhanced Hypercube Network The work of Hsu, Yew, and Zhu [HsY87] focuses on the introduction of a new interconnection network structure, the *Block-shuffled Hypercube* (BSH). It is proposed that the BSH implementation may be able to replace the hypercube network with minimum hardware changes. The interconnection scheme of the BSH is based on the shuffle-exchange interconnection method of [Law75]. As an enhancement to the hypercube interconnection network, [HsY87] show that the hardware requirements of the BSH network are substantially less than those required by the hypercube. Using an message generation approach identical to [AbP86], it is shown that the BSH network outperforms the hypercube interconnection network in terms of message delays and hardware costs. Based on the reasoning of [AbP86], the authors further conclude that the BSH network also outperforms the multistage cube networks of similar network size.

3.7 *Summary*

In this chapter, the state of performance modeling and analysis of interconnection networks was examined. The use of computer simulations to model interconnection networks provide a low cost and timely method for determining the feasibility of a system(s) in question. VLSI comparisons of interconnection networks prove invaluable as the technology is rapidly progressing to meet the needs of systems which require processing units on the order of tens of thousands.

The reviews of the performance modeling and analysis of interconnection networks presented above reveal that much is yet to be learned from these types of analysis. First, the present studies are limited in the scope of their comparisons. Normally, one type of network is compared against the next under simplifying and constraining assumptions. There needs to be comparisons of classes of networks in a broader sense. The works by [Dal86] and [AbP86] present examples of the limited nature of performance comparisons. While Dally contends that low-dimensional networks perform better that do high-dimensional networks such as the multistage cubes, this contention may not be totally correct. One of the main points presented by [DaS86] was the packet offset time or time differential between packet generation and packet presentation to the network. This is an important point to consider. If the processing elements take orders of magnitude longer to present the packet to the network than the packet will take to traverse the network, the bottleneck has now shifted from the network to the processing elements. If this is the case, reexamination of performance comparisons, such as the ones presented above, must be researched. In Dally's work, this time differential is not addressed. Additionally, the work of Abraham and Padmanabhan is limited in its scope. In their comparison of the direct binary n-cube with indirect binary n-cubes, the size of the switching elements was limited to 8-by-8. While this may have been a sufficient limiting size to validate their research for small size crossbar switches, work by [Szy86] has shown that with present VLSI techniques, crossbar sizes of 32-by-32 are now realizable. Further work is necessary to determine what effects the increased crossbar switch sizes will have on the networks examined.

The information contained in this chapter is used as a base for further research in the parallel processing systems environment. Specifically, the following chapters present the methods used and results obtained in the comparison of three interconnection networks whose sizes range from 64 to 1024 processing elements. The interconnection networks to be modeled for evaluation are the single stage cube, the multistage cube and the Illiac IV networks.

4. Interconnection Network Modeling

4.1 Introduction

This chapter presents the methodology used in the modeling and simulation of the multistage cube networks, the single-stage cube networks, and the Illiac IV mesh network. A discussion of network modeling via computer simulations is presented in Section 4.2. In this section, an introduction to the SLAM II simulation language is presented along with the benefits it lends to the modeler. This discussion is followed by Section 4.3, which defines the network internal and external environments and the operating assumptions which have been used to facilitate this investigation. The formulation of the network models are outlined in Section 4.4 with an in-depth discussion of the channel allocation-deallocation problem. The approaches used in designing and simulating the multistage cube networks, the single-stage cube networks, and the Illiac IV mesh networks are explained in Sections 4.5, 4.6, and 4.7 respectively.

4.2 SLAM II and Interconnection Network Modeling

In recent years, network modeling via computer simulation has been implemented primarily in high-ordered languages such as C and Fortran. These simulations, such as [Ove82], have been comprised of thousands of lines of source code which adds to the complexity of the modeling effort and the overall time required to complete the effort. To reduce, and possibly eliminate the necessity of high order language simulations, Pritsker and Associates, Inc. [Pri86] developed a Fortran-based language, *S*imulation *L*anguage for *A*lternative *M*odeling (SLAM) which is ideally suited for network simulations. The compactness and completeness of its code makes SLAM modeling desirable. For this reason and due to the expertise available for designer's questions, SLAM was chosen as the tool for use in modeling the interconnection networks of this investigation. A brief description of the SLAM network language is provided to assist in the understanding of the models to be presented in the latter sections of this chapter.

In the modeling of a network, the SLAM simulation consists of two parts: the control statements and the network description. The control statements provide options to the user for determining the initial states, any modifications to the simulation and when and how to terminate the simulation. The network description is the SLAM code which represents the modeler's interpretation of the actual network process. SLAM provides to the u_er a set of 23 network statements which allow for in-depth simulations ranging from complex computer networks [Gar85], to computer interconnection networks [AlH86].

SLAM provides the *entity* which is used to model a message that will flow through the network in a store-and-forward manner. Each entity can have associated with it a set of *attributes* which are used to distinguish one entity from the other. These attributes can be assigned values which represent source addresses, destination addresses, message lengths, or other user defined values. The *file* is used to represent resources such as channels or memory modules as well as queues which store groups of entities. The basic concept of the SLAM model is to have the entity(s) generated at some prescribed interarrival rate and then flow through the network, following the routes determined by the designer. Each entity which enters the network will be also be terminated and removed from the network. Upon termination, statistics associated with the entity may be collected when specified.

The statistical results of the simulation are provided for in SLAM via a summary report. The summary report includes statistics on the files, activities, and/or variables of the model. The summary report is the primary output of a SLAM simulation. Additional information concerning the simulation can be obtained from echo reports, and trace reports. Echo reports reflect the data input and the initial values set prior to execution. The trace reports are primarily used as debugging and validation tools for the model. The trace report provides a snap-shot view of the network at each instance of time in which an event is scheduled to occur.

4.3 Network Operating Assumptions

To facilitate this investigation of the mesh, the single stage cube and the multistage cube interconnection networks, certain assumptions and operating conditions for the networks must be established. Based on the operating assumptions used in previous research [DiJ81], and from the discussions in Chapters 2 and 3, the operating assumptions and conditions for the network simulations are described below.

1. Each of the networks to be modeled are assumed to be operating in a MIMD environment.
2. A PE-to-PE architecture is assumed.
3. Packet switching is used as the method for inter-PE communications. Message buffers are employed at the switches for the storing and forwarding of packets to and from the switches.
4. Message interarrival times are assumed to be Poisson processes.
5. Generation of source and destination PE addresses are uniformly distributed over the range of values specified by the number of PEs in the network.
6. Messages are assumed to be single packets in length.
7. The unit of measure for determining the average message delay and throughput of the network is the *packet cycle time*. This is the time that it takes a packet to move from the front of a queue, through its corresponding switch, and arrive at the queue associated with the next point (switch) in its routing scheme. For all simulations, the packet cycle time is normalized to 1 unit.
8. Network outputs can process messages faster than the messages can be generated. This insures that the output device will not be a bottleneck.

9. Message buffers are infinite in length. The rate in which packets arrive at the input queues is controlled by a Poisson process. This arrival rate determines the relative load on the network along with the average and maximum size of the crossbar switch queues necessary to store the arriving packets for a given rate. Packets entering these buffers are transmitted on a first-come-first-serve basis (FIFO).

4.4 Formulation of Network Models

The focus of this investigation is to determine how two network performance parameters, message delay and network memory costs compare for dissimilar interconnection networks. Specifically, by comparing these parameters for the mesh, single stage cube and multistage cube interconnection networks, the desirability of one network over another can be determined for an arbitrary network size at various loads. Recalling from Chapters 2 and 3, the switching element most commonly used for connecting autonomous processors is the crossbar switch. Since current technology allows for implementations of crossbar switches to be as large as 32-by-32 [Szy86], modeling an interconnection network using crossbar switches appears feasible and realistic. Using the crossbar switch as a focal point, the networks to be investigated have a common starting point in the modeling effort.

4.4.1 The Allocation Problem As aforementioned, the tool used to solve this investigation is SLAM II [Pri86]. This versatile language allows the user to design models with the intricacy and diversity limited mainly to the designer's abilities. To support the basic language, SLAM allows the designer to write code in Fortran to perform tasks that may not be normally supportable.

The initial approach to modeling the three interconnection networks is to examine a relatively small network with the smallest crossbar switch available. A 64-PE multistage cube network which uses 2-by-2 crossbar switches is chosen as the starting point of the modeling effort. The first obstacle to overcome is the instantiation of code for modeling a set of parallel and autonomous processes. Two methods existed: for an N-PE system, instantiate the create process for messages N times, which would cause a massive duplication of code; or use an interarrival rate which would allow the N processes to be simulated using one message creation node. The latter technique is chosen. By allowing the interarrival rate to be a Poisson process, mathematically the aggregate of the interarrival rate can be decomposed to provide the rate necessary to model the N parallel processes.

Using this type of interarrival rate provides the possibility to model a network of arbitrary size by modeling how one entity (message) flows through the network. This further allows for a simulation such as [Ove82], to be reduced from thousands of lines of code to less than one hundred. By resolving the parallel code instantiation problem, this investigation proceeds by examining the next major obstacle, modeling the entry and removal of entities (packets) from the crossbar switch input queues. This process turns out to be one of most time consuming and thought provoking aspects of this investigation.

To appreciate the solution to the problem of entity entry and removal from a switch queue, it is necessary to explain the trials and errors associated with obtaining the solution. In the networks that are to be modeled, three important pieces of information are absolutely necessary: where the packet was coming from; to which queue would the packet be routed; and to which outgoing channel would the packet be routed. Using mathematical relationships to be discussed in the sections related to each particular network, the above three requirements are obtainable. Still to be resolved is the allocation of an available channel and the reallocation the same channel when it is applicable.

Understanding the allocation problem encountered requires additional information on how the SLAM processor works and description of the SLAM AWAIT node, RESOURCE block and FREE node. In SLAM, all events that occur associated with entity (packet), from creation time to final termination, are scheduled on an event calendar. This event calendar is a doubly linked list with the capability of being scanned forward and backward. V'hen an entity is created, statistics are kept on that entity as it flows through the network. Any waiting that the entity encounters (i.e., queueing time) is also kept in the statistics associated with the entity and the file corresponding to the queue the entity entered. The event calendar is incremented only through the use of an *activity* which specifies the duration of a particular event such as the packet cycle time.

Critical to this investigation is the control of the communication channels between crossbar switches. Physical implementations allow one packet to be using a channel at any instance of time. Subsequent requests for the channel will be granted only when the channel is free. Packets requesting a busy channel must be queued or dropped from the system in accordance with the packet disposition algorithms described in Chapter 3. Initially, the SLAM RESOURCE block appears to be the best method for modeling the channels connecting the crossbar switches. A RESOURCE block allows the designer to specify the capacity of the resource and which files (queues), if any, should be polled when the resource is available. Used in conjunction with a RESOURCE are the AWAIT and FREE nodes. As an entity flows through the network, it can enter an AWAIT node requesting access to a particular RESOURCE. If the RESOURCE is available, the entity seizes a specified number of units of the requested RESOURCE (in the case of the networks to be simulated one unit of a particular resource was the capacity) and proceeds to a service activity which increments the event calendar and the entity's time in the system. The service activity specifies the duration of the event simulated. Upon completion of the service activity, the entity flows into a FREE node which deallocates a specified number of units of the seized RESOURCE. Once the RESOURCE has been deallocated, SLAM automatically checks if any requests for the RESOURCE have not been filled. This is done by keeping a list of what files (queues) wanted access to the RESOURCE when it was busy. If requests are waiting, this list is used to determine which entity will be allowed to allocate the RESOURCE next. SLAM then scans the files associated with the resource declaration in the order specified by the declaration statement. As an example, consider the following code.

```
RESOURCE/1,CHAN1(1),113,121;

                     .

                     .

                     .

                     .

AWAIT(QUE=65,128),CHAN/1;
ACT/1,1;

                     .

                     .

                     .

                     .

FREE,CHAN/1;
```

The above code declares a resource named CHAN1, which has a capacity of one unit and a SLAM file number equal to 1. Two files (queues) associated with CHAN1 are 113 and 121. This declaration tells SLAM that files 113 and 121 are the files to examine *in that order* when CHAN1 is requested. Previous to an entity flowing into the AWAIT node, values are assigned to QUE and CHAN to correspond with the input queue of a crossbar switching element and the output channel from the crossbar switch to the next input queue in the entity's routing path. Assume that QUE = 113, and CHAN = 1 (indicating CHAN1 is requested). When an entity flows into the AWAIT node, if CHAN1 is busy then the entity is stored in file 113. If CHAN1 is free, then the entity seizes CHAN1, which locks out subsequent requests and then performs the following activity ACT/1,1. This activity tells the SLAM processor that the next event for this particular entity is to be scheduled one time unit in the future. After one time unit has expired the entity flows into the FREE node where the deallocation and reallocation of CHAN1 is determined. If additional requests for CHAN1 are present, file 113 is first checked and then 121 for requestors. As long as requestors exist in file 113, file 121 will never be checked. For the purposes of this investigation, the sequential manner in which the SLAM processor polled the files associated with a RESOURCE proves inadequate. This is due, in large part, to the physical operation of the crossbar switch.

In the physical operation of the crossbar switch, one method of determining channel allocation is the longest-waiting-first scheme. In each of the FIFO input queues, the first packet is polled to determine if the packet desires the free channel. The oldest packet of the ones waiting for a particular channel is then chosen for removal from its queue and allowed to traverse the channel, blocking others waiting until its traversal is complete. This is the desired approach for implementation in the ALLOC subroutine. Conferring with Pritsker and Associates Technical Consultants [Pri87], assurance was given that this type of implementation was possible but that presently, there did not exist documentation of attempts to perform this complex of an allocation scheme using the ALLOC routine and RESOURCEs.

At each crossbar switch, the number of input queues to the switch are dependent upon the number of PEs in the network and the type of interconnection network modeled. To model the physical operation of allocating a channel from one of many input queues, an alternative to the sequential method of file polling is sought. The SLAM processor provides to the user the possibility for user defined allocation schemes via the ALLOC subroutine. This Fortran subroutine allows for complex allocation schemes which are normally not supported by the SLAM processor.

Pursuing the channel allocation scheme using the ALLOC subroutine requires that this routine be called each time one of the following two cases occurs. First, ALLOC is called when an entity arrives at an AWAIT node and secondly, when an entity arrives at a FREE node. This adds extra burdens on the designer to insure that the correct logic is executed when ALLOC is called. The implementation of the logic associated with the call to ALLOC at an AWAIT node is relatively straight forward. If the channel is free and the queue in which the arriving entity is associated with is empty, let the entity seize the RESOURCE and pass through the AWAIT node. If the RESOURCE is busy or the input queue is not empty, then file the incoming entity in the appropriate queue. The complex portion of this allocation scheme occurs at the FREE node.

At the FREE node, ALLOC is called to determine if an allocation can be made. For this portion of ALLOC, the determination of the oldest entity waiting for the channel to be freed is to be made. While the designer is allowed to manipulate the SLAM files to extract the oldest waiting entity, the unsolvable part of this allocation comes from the inherent limitations of SLAM and the sequential polling of the files associated with the RESOURCE declaration. For an entity to flow through the network, the entity's attributes must reside in the ATRIB buffer. This buffer contains the attributes of the entity which is currently traversing the network. These attributes remain in the ATRIB buffer until the entity's traversal is forced to stop due to a QUEUE, AWAIT, or FREE node. At any of these nodes, the attributes of the arriving entity are filed in the appropriate file with the ATRIB buffer assuming the attributes of the entity whose rank is one and resides in the first nonempty file associated with the corresponding RESOURCE block. The fatal error with attempting to use ALLOC to solve the allocation problem is that if the oldest entity that requests the channel is not the same entity as the one placed in the ATRIB buffer, then one of two situations can occur. First, any attempt to place the oldest entity in the ATRIB buffer results in the loss of the entity whose attributes previously resided in the ATRIB buffer. Second, realizing that the oldest entity and the entity in the ATRIB buffer do not match, the designer of ALLOC informs the SLAM processor that no allocation can be made. This second situation causes the SLAM processor to attempt another allocation by calling ALLOC a subsequent time. If the entities do not match, the processor attempts to free the RESOURCE associated with the entity in the ATRIB buffer causing an error to occur in deallocating a RESOURCE that had not been allocated. Exhausting possible "tricks" to fool the SLAM processor into allocating the oldest entity via ALLOC, and due in large part to the time factor involved with this investigation, an entirely different approach is considered in attempting to solve the allocation-deallocation problem.

4.4.2 The Allocation Solution Abandoning the attempt to model the networks via the use of RESOURCEs, the method pursued, with the solution following is to discretely code the actions of the AWAIT and FREE nodes in an EVENT node. The EVENT node is provided in SLAM to allow free access to file manipulation with the exclusion of the event calendar file. When an entity flows into an EVENT node, its attributes remain in the ATRIB buffer through the entire operation of the logic associated with the EVENT node. This aids in the solution of the allocation-deallocation problem since the switch input queues and outgoing channels are attribute-based calculations which are carried by the entity as it flows through the network. Used in conjunction with the EVENT node, is the ENTER node, which allows for selective entity entry in to the network. The basic function of the EVENT node is to determine the oldest waiting entity and allow for its placement into the network by the ENTER node. The algorithm flowchart for the EVENT node is shown in Figure 4.1.

In place of the RESOURCE block used to model the inter-switch box communication channels, are attribute-based ACTIVITIES. Each ACTIVITY is uniquely defined by an integer number ranging from one to the number of channels in the network. Much like the RESOURCE, an ACTIVITY can be accessed and held by only one entity for the duration specified by the ACTIVITY. Unlike the RESOURCE, the ACTIVITY has no automatic reallocation process; it is released at the end of the time interval specified in the ACTIVITY statement and is free to be accessed by any entity requesting it. For this investigation, using the ACTIVITY to model the channel proves to be the ideal solution.

4.5 The Multistage Cube Network Model

The multistage cube interconnection network was chosen as the first network to be modeled due to the need to compare and validate the simulation results with previously published research [DiJ81, KrS83]. These results will be discussed in detail in Chapter 5.

Throughout the modeling of the three chosen interconnection networks, mathematical relationships were sought for the appropriate network parameters. Achieving these types of relationships allowed for the source code to be compact and easy to follow logically. Along with the use of mathematical relationships, the use of Poisson interarrival rates allowed the SLAM source code needed to simulate an arbitrarily sized multistage cube network to be less than 50 lines. Figure 4.2 shows the SLAM graphic representation of the multistage cube network.

Associated with each generated entity (message) is a set of nine attributes which distinguish the network entities. These attributes are defined as follows:

1. ATRIB(1) : the time of entry into the system
2. ATRIB(2) alias SRCE : the source address
3. ATRIB(3) alias DEST : the destination address
4. ATRIB(4) alias STAGE : the present stage
5. ATRIB(5) alias QUE : the input queue of a crossbar switch
6. ATRIB(6) alias CHAN : the outgoing channel number
7. ATRIB(7) alias BITS : the indicator bits that are passed to the EVENT node to determine the queues that should be scanned at a particular crossbar switch
8. ATRIB(8) alias INTIME : the time an entity enters the EVENT node
9. ATRIB(9) alias DUMMY : dummy flag attribute for entering the EVENT node

Also associated with the network simulation are three global variables. These variables are used throughout the simulation with their values remaining static.

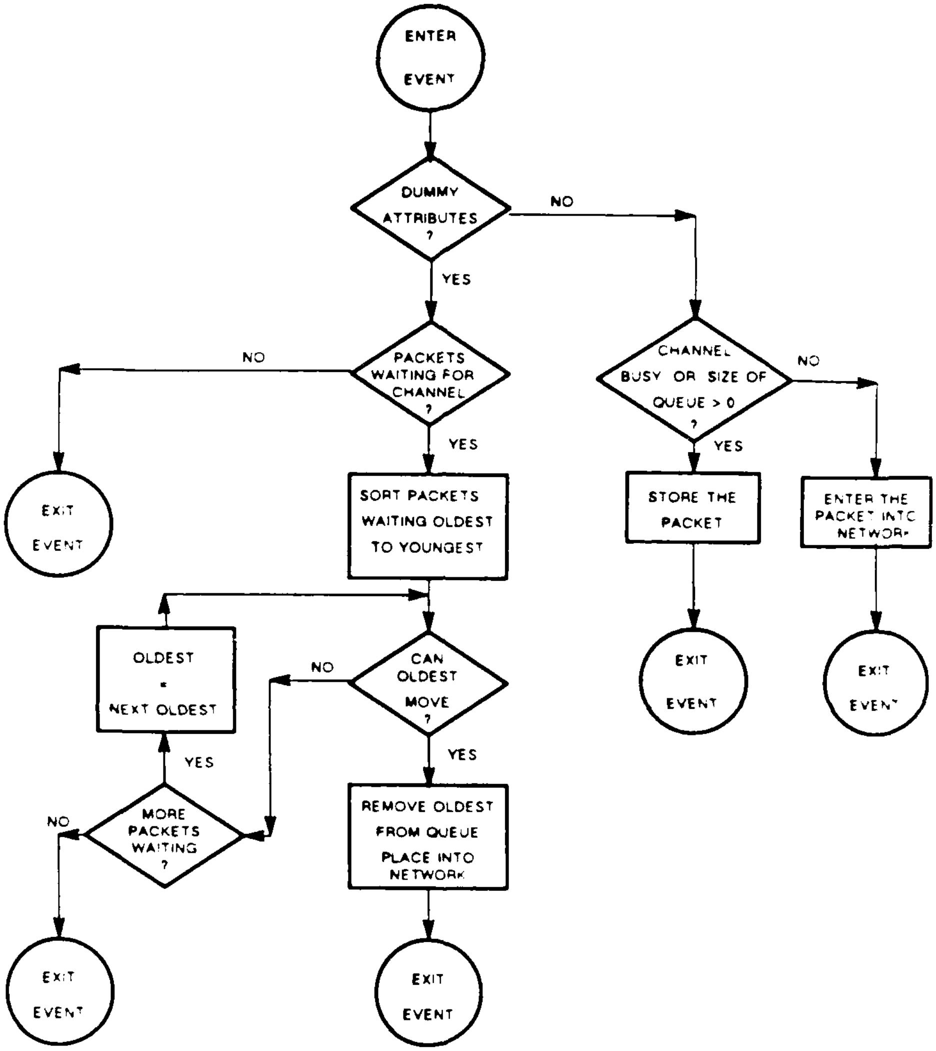


Figure 4.1. EVENT node flowchart for allocation-deallocation process

1. XX(1) : the number of input queues at a crossbar switch
2. XX(2) : the number of routing tag bits that must be examined for each pass in USERF(1)
3. XX(3) : number of nodes in the network

Each simulation begins with the creation of a packet at time zero and thereafter at a rate specified by the Poisson interarrival times. Upon creation, a packet is assigned a source-destination address pair based on an uniform distribution whose range of values is restricted by the number of nodes in the network. Next, the packet's appropriate attributes are assigned a value for the present stage and the initial input queue. The packet continues to flow into a sequence of statements which assign and reassign values to the outgoing channel, the present stage and the input queue to the next stage. This sequence constitutes a loop which steps the packet through the network toward its destination. The packet exits the loop when it reaches its destination. At this point, statistics are gathered on the packet's time in the system.

Within the loop is the packet flow control, the EVENT node. The EVENT node controls the number of stages that a packet is allowed to flow through unrestricted before it must be stopped and filed in the appropriate queue.

4.5.1 Packet Routing in the Multistage Cube Model As described in Chapter 2, the packet routing in a multistage cube interconnection network is deterministic. For each stage in the network, address lines are grouped at switch boxes according to the $Cube_i$ function. For example, at stage 3, of a 16-node network, address lines 0 and 8 are grouped together in a 2-by-2 crossbar switch. The routing to the next stage is dependent upon the source-destination pair and the routing algorithm used (XOR or destination routing).

Of particular importance in determining the outgoing channel are the current stage number, the bits to be examined in the address, the size of the crossbar switch (i.e., 4 for a 4-by-4), and the source-destination address pair. Implementing the rout-

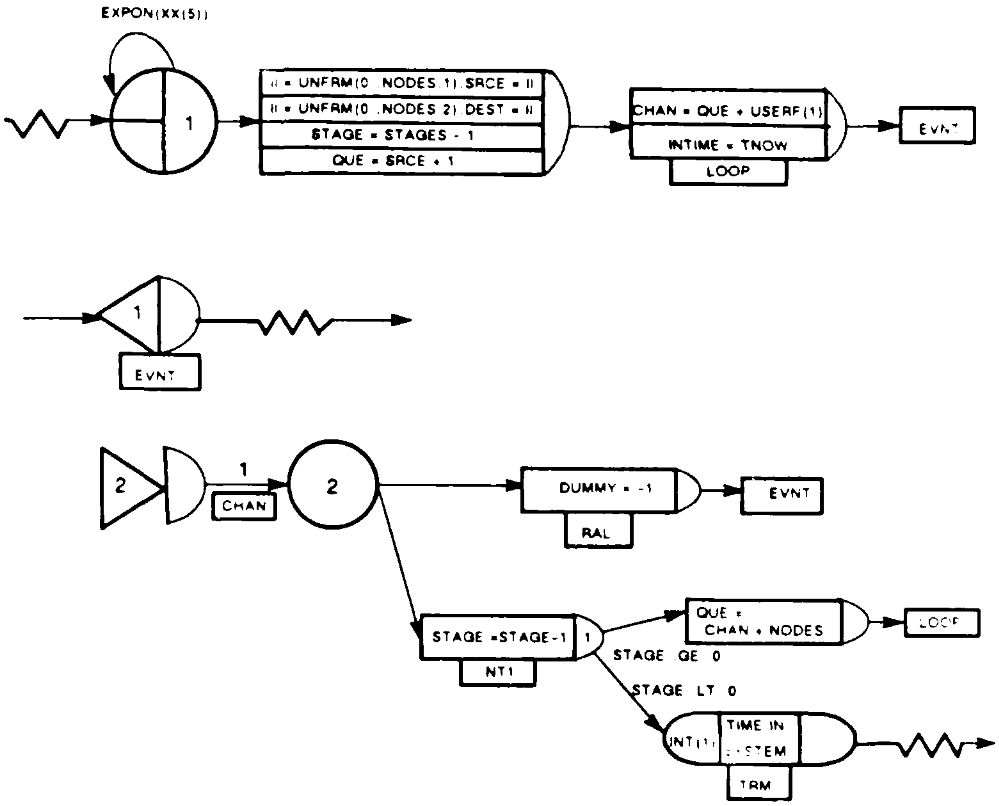


Figure 4.2. SLAM graphic representation of a multistage cube network

ing algorithm in SLAM requires that the algorithm be discretely coded in Fortran. SLAM supplies to the user, a function USERF, which allows for decision modeling which is not supported by the basic constructs. Using USERF and ATRIB(5), the equation for the outgoing channel is given by:

$$CHAN = QUE + USERF(1) \tag{4.1}$$

USERF(1) determines the relative position of the outgoing channel based on the relative position of the input queue where the packet is currently residing and the destination address. These relative positions are determined by examining particular bits of the source and of the destination addresses. The range of values for the relative positions is limited by the size of the crossbar switch. Once the relative positions have been determined, USERF returns a value to CHAN based on the following equation.

$$USERF = (destination_{rp} - source_{rp}) \times (size^{stage}) \tag{4.2}$$

where

$destination_{rp}$ is the relative position of the destination address.

$source_{rp}$ is the relative position of the source address.

$size$ is the crossbar switch size.

$stage$ is the current stage number.

Once the outgoing channel is determined, the calculation of the input queue at the next stage can be made. Using an internal numbering relationship, the input queue number at the next stage is equal to the outgoing channel number from the previous stage. By numbering in this manner, only one calculation has to be made for any channel-queue pair.

4.6 The Single Stage Cube Network Model

Using an approach similar to the one used in modeling the multistage cube interconnection network, the simulation code for the single stage cube interconnection network of arbitrary size is compact and modular. The SLAM graphic representation is shown in Figure 4.3.

As in the multistage cube interconnection network, the single stage cube model has nine attributes associated with each entity (packet) that is placed in the network. These attributes are:

1. ATRIB(1) : the time of entry into the system
2. ATRIB(2) alias SRCE : the source address
3. ATRIB(3) alias DEST : the destination address
4. ATRIB(4) alias BOX : crossbar switch number
5. ATRIB(5) alias PSRCE : previous source number
6. ATRIB(6) alias CHAN : outgoing channel number
7. ATRIB(7) alias QUE : the input queue number
8. ATRIB(8) alias INTIME : time entity enters EVENT
9. ATRIB(9) alias DUMMY : dummy flag for EVENT node

Also, four global variables are needed for the simulations. These variables remain static throughout the simulations and are defined as follows:

1. XX(1) : the network size
2. XX(2) : the size of the crossbar switch
3. XX(3) : the number of channels in the network
4. XX(4) : bit counter for USERF(2) and USERF(4)

The creation of packets into the network is controlled by a Poisson process. Once created, packets are assigned source-destination addresses based on a uniform distribution over the range of the number of processor in the network. The assignment of the initial input queue follows the source-destination assignments. Following

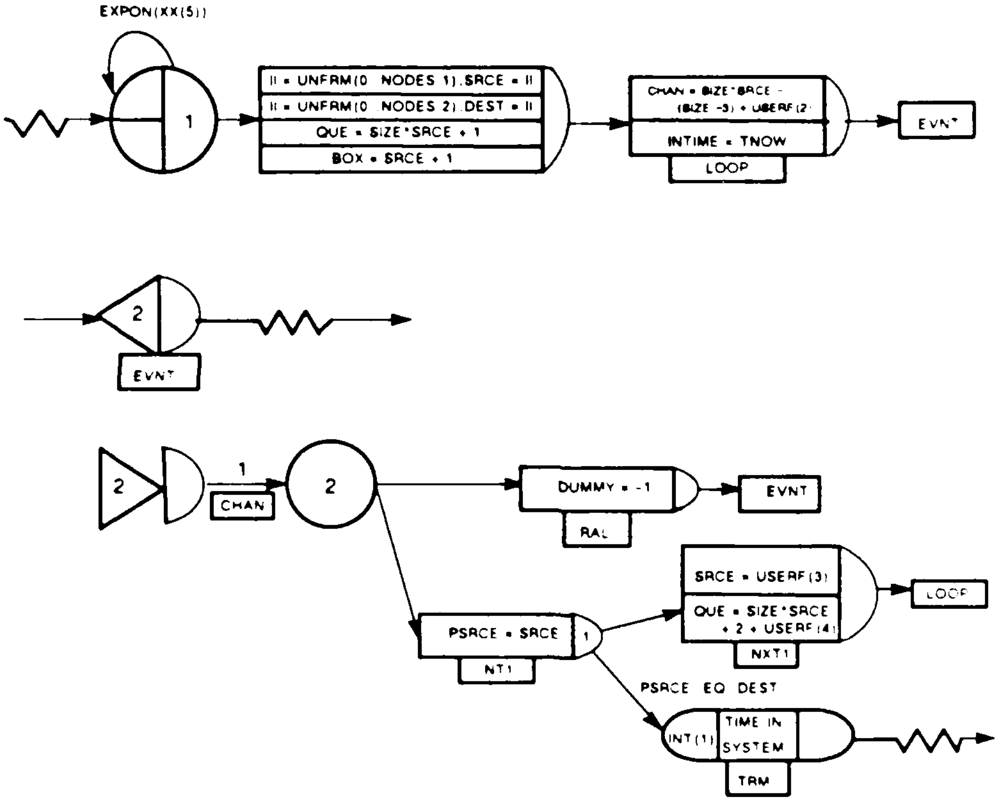


Figure 4.3. SLAM II representation of a single stage cube network.

these initial assignments, the packet enters a section of code which steps the packet through the network toward its destination node. In this section, the outgoing channel, the next intermediate source and the next input queue are determined. How far an entity proceeds through the network is determined in the EVENT node described previously in Section 4.4. When an entity reaches its destination, statistics are collected on the total time the entity spent in the network. The entity is then terminated and removed from the system.

4.6.1 Packet Routing in the Single Stage Cube Model Recall from Chapter 2, that in an m-dimension single stage cube interconnection network, each processor is connected to m other processors by the Cube$_i$ interconnection function. This requires that the crossbar switch at each node be of size $m + 1$. Once a packet is placed at a node, a routing algorithm must be employed to move the packet to the next node in the routing path. The Intel iPSC uses a "hardwired" method [Int86] of routing using one of seven local area network (LAN) channels to move the packet from one node to the next. A particular output channel is dependent upon the current node and the next node in which the packet is to be routed.

Mathematically, the routing algorithm of the iPSC performs an exclusive-or of the source-destination addresses and then scans the resultant bits until it encounters a logic 1. The LAN channel to be used is the bit position of the first 1 encountered when scanning from least significant bit to most significant bit. Transmitting the packet along the chosen LAN channel places the packet at a new source node, where the above process is repeated until the new source is equal to the final destination address.

The above routing algorithm is used in the simulations of the single stage cube interconnection network models. Besides having to know the communications channel, knowing the new source and the new input queue are also needed. These parameters are given by the following equations:

$$CHAN = SIZE \times SRCE - (SIZE - 3) + USERF(2) \qquad (4.3)$$

$$SRCE = USERF(3) \qquad (4.4)$$

$$QUE = SIZE \times SRCE + 2 + USERF(4) \qquad (4.5)$$

where SIZE is the size of the crossbar switch

USERF(2), USERF(3), and USERF(4) are user written Fortran functions.

USERF(2) returns a value between -1 and SIZE−2 based on the bitwise exclusive or of the source-destination addresses and the occurrence of a 1 in bit position *i*. When the source and destination addresses are the same, a -1 is returned by USERF(2) indicating that the packet has reached its destination. If the source address does not equal the destination address, a positive value corresponding to the bit position of the first 1 encountered is returned by USERF(2).

USERF(3) determines the next node in the routing sequence. An approach similar to the one used in USERF(2) is used in USERF(3). A bitwise exclusive or is performed on the source-destination address pair with the bit position of the first 1 encountered recorded. The next node is then determined by one of the following two equations

$$NXTNODE = source + 2^{bp} \qquad source < destination \qquad (4.6)$$

$$NXTNODE = source - 2^{bp} \qquad source > destination \qquad (4.7)$$

where *bp* is the bit position of the first 1 encountered.

USERF(4), in part, calculates the input queue at the new source. The approach is identical to USERF(2) except USERF(4) returns a value between 0 and

SIZE−2. These three USERF functions used in conjunction with the current entity's attributes, give mathematical relationships for the parameters needed in simulating the single stage cube network. This allows for the simulation of arbitrary sized networks with little change necessary to the SLAM source code and no change to the Fortran functions or subroutines.

4.7 The Mesh Network Model

The third interconnection network modeled in this investigation is the Illiac IV mesh type network. The Illiac IV differs from a mesh network in that the processing elements that lie on the edges of the network are connected to other edge processing elements in a "wrap-around" manner described by the four Illiac IV interconnection functions defined in Chapter 2. As with the multistage cube interconnection network and the single stage cube interconnection network, the Illiac IV network uses crossbar switches for the interconnection of processing elements. Where the Illiac IV network differs from the other two networks, is in the size of the switch. In the Illiac IV network, the size of the crossbar switch is fixed at 5-by-5 for any size network. The size is fixed due to the structure of the mesh type network, where any processing element is directly connected to four neighboring processing elements.

The SLAM modeling approach for the Illiac IV interconnection network is identical to the approach used in modeling the single stage cube interconnection network. All entity attributes and global XX variables are defined in the same manner as in the single stage cube model. The two interconnection networks differ in their respective interconnection function implementations.

4.7.1 Packet Routing in the Illiac IV Model. The SLAM implementation of the packet routing in the Illiac IV network is performed, in part, by the use of three user written Fortran functions, USERF(5), USERF(6), and USERF(7). These functions return values to the following three equations to determine the outgoing channel, the next source node, and the input queue number at the next source node.

$$CHAN = SIZE \times SRCE - (SIZE - 3) + USERF(6) \qquad (4.8)$$

$$SRCE = USERF(5) \qquad (4.9)$$

$$QUE = SIZE \times SRCE + 2 + USERF(7) \qquad (4.10)$$

The user written function USERF(5) determines the next node in the routing sequence. This is done by first performing the four Illiac IV interconnection functions to determine if the current node is directly connected to the destination node. If a direct connection exists, then the new source node assumes the value of the destination node address. If the current node is not directly connected to the destination node, then USERF(5) returns a value for the new source node which is equal to one of the four neighboring nodes. The neighboring node chosen is determined from a set of rules which finds the minimum difference between the destination node and each of the four neighboring nodes of the current source node. This minimum difference insures that the routing path from any source to any destination node is the shortest path. In the event that multiple shortest paths exist, the first shortest path encountered is the one taken.

The outgoing channel equation is defined above and uses USERF(6) to calculate the relative position of the outgoing channel at the crossbar switch. Identical to USERF(5), USERF(6) first determines if a direct connection exists between source and destination nodes. If the direct connection exists, USERF(6) returns -1 to indicate that the packet has reached its destination. If the current node is not equal to the destination node, USERF(6) returns a value between 0 and 3 inclusive which indicates the relative position of the outgoing channel at the crossbar switch.

USERF(7) is used to assist in the calculation of the input queue number at the next source node's crossbar switch. Its function is identical to USERF(6) except the range of values returned is limited from 0 to 3 inclusive. USERF(7) gives the relative position of the input queue at the next source's crossbar switch.

4.8 Summary

In this chapter, the methodology used in formulating the multistage cube the single stage cube and the Illiac IV interconnection networks was discussed in detail. For each of the networks examined, a unique SLAM model was designed and implemented. The problems encountered in initial attempts to model these networks were also discussed to lend insight to future such investigations. These models will be used to determine the performance of one network versus the other when considering the average time that a packet spends in the network. This performance measure is discussed in detail in Chapter 5.

5. Network Simulation, Validation, and Performance Comparisons

5.1 Introduction

This chapter presents the simulation, validation, and comparisons of the three interconnection networks modeled in Chapter 4. Section 5.2 presents a discussion of the time and machines required to perform the network simulations along with the techniques used to ensure valid statistical representations of the network models. The network validation procedures are discussed in Section 5.3. The delay characteristics of each network are presented in Section 5.4. In this section, individual network delay characteristics are examined. The delay characteristics of the three networks are compared and evaluated in Section 5.5. Network buffer requirements along with the memory costs of each network are also presented in Section 5.5.

5.2 Network Model Simulations

This investigation consists of four phases: the network modeling, the network simulations, the network validations, and the performance comparisons of the networks modeled. Having discussed phase one in Chapter 4, phases two, three, and four remain. This section presents the approach taken in simulating the three interconnection network models.

In attempting any large scale simulation, one of the major concerns to be considered is if the facilities are capable of supporting the simulation. This investigation is believed to be the first at AFIT to attempt such a large scale simulation using the simulation language SLAM. In modeling the interconnection networks chosen, a wide range of memory requirements are needed. Also, the processing speed of the machine is critical in a time-constrained investigation such as this one. In addition, the anticipated loading on the system due to other users is of critical interest. All of these factors are issues that must be researched prior to attempting the simulations

For the network models to be simulated in this investigation, only one machine is available to best handle the issues raised above. The ICC, an Elxsi super minicomputer-class machine, provides the real and virtual memory requirement, processing speed and system load capable of supporting simulations where tens of thousands of SLAM files are used. The ICC is a duai processor machine which has processor speeds of 6 and 12 mips (million instructions per second). Virtual and real memory storage is capable of handling concurrent processes on the order of 50 Megabytes. The processing speeds of the icc are 7 and 15 times faster than the Vax-class machines previously used for SLAM simulations. This results in simulation turnaround times being reduced from cpu hours to cpu minutes.

To get a "feel" for the size of the simulations to be performed, it is necessary to give a few of the parameters that are specific to a given simulation. To simulate any job in SLAM, certain parameters must be specified prior to the simulation. For example, associated with a SLAM job is the Fortran subroutine MAIN. In MAIN, the network size parameters are defined and memory allocated according to these definitions. These parameters range from the arrays which contain the attributes of every entity that enters the network, to arrays which specify the maximum number of files, activities, and resources allowed by the latest configuration of the SLAM executable code. One of the most important set of arrays are the NSET/QSET arrays. The NSET/QSET arrays specify the array sizes that must be "set aside" to store the attributes of the entities which enter the network. Refer to [Pri86] for a in-depth explanation of NSET/QSET and its value derivations. For the largest networks to be simulated, the NSET/QSET arrays are set to 8,000,000. This in turn causes the network model's executable code to be approximately 35 Megabytes in size. Also of interest is the execution time required to simulate various size networks. For networks comprised of 64 PEs, approximate simulation times are on the order of 15 cpu minutes. Large networks, simulating 1024 PEs require cpu times on the order of 40 cpu hours. The turnaround times of these network simulations are, to a large part, dependent on the load placed on the system by other users. Large jobs take on average, 1 to $1\frac{1}{2}$ calendar days per simulation run.

Defining the network parameters to be investigated proves to be the first step to be taken in simulating the interconnection network models. For this investigation, two network parameters are focussed upon: the average message delay and the buffer memory cost requirements of each network. The average message delay is defined as the time required by a message to traverse the network from input PE to output PE. The buffer memory cost is the product of the total number of buffers in the network, the maximum buffer length (length required to ensure that 99% of the time that the network is in operation, this length will not be exceeded), and the unit cost per memory size (assumed to be constant at one unit for this investigation).

Upon defining the network parameters of interest, message delay curves for the three networks must be generated. The generation of these curves proved to be time consuming and tedious. For each of the curves to be generated, multiple simulation runs are associated with each discrete point on the delay versus network loading curve. With each new iteration of the simulation, new seed values for the random number generators must be given. These random number generators are used in the generation of PE source-destination address pairs. Determining the number of simulation runs associated with each data point is dependent upon the desired accuracy and degree of confidence in the mean average message delay value obtained from the multiple runs. In addition to determining the number of simulation runs necessary for a given point, the number of points that are required to accurately reflect the average message delay curve characteristics must be determined. These two decision factors must be carefully researched, through pilot simulation runs, to insure an accurate representation of the networks' delay characteristics are obtained.

Each delay curve that is obtained consists of five data points. This number is chosen due to the reasons that follow. First, observed from the pilot simulation runs, the average message delays for the networks remain approximately constant for light to medium network loading (the loading factor is discussed below). This allows for a minimal number of data points to be examined in these loading ranges. Thus, the emphasis of data point distribution can be placed in the area of the "knee" of the message delay curve. The "knee" is the area of the curve where queueing delays become more prevalent but not to include the portion of the curve where the network is in saturation. Pilot simulation runs are required for each data point. These runs are necessary to ascertain the steady-state delay value for a given load.

The steady-state condition of a system is when the system's operating characteristics (in this case, the message delay) do not change over time. To reach steady-state in a system, an initial "warm-up" period is needed to allow the system to reach the point of normal operation. SLAM has a built-in construct which aids in determining the steady-state of a system. Using the MONTR, SUMRY statement, a "snap-shot" of the system parameters of interest can be obtained for any desired time interval. The steady-state condition is determined by analyzing the intermediate summary reports produced by the MONTR, SUMRY statement. Once the determination of the time required by the system to reach steady-state has been made, a second SLAM construct can be used to clear the statistical values of the network that have been kept prior to the steady-state condition. The MONTR, CLEAR statement allows the designer to clear the SLAM statistical arrays at a specified time into the simulation.

Simulations of the network models require that each network be operating in the steady-state region prior to obtaining the delay curve characteristics. Therefore, each network simulated requires the iterative process of performing time "snap-shots" of the system and clearing the statistical arrays once the steady-state simulation time has been determined. This requires exorbitant amounts of CPU time to determine the network steady-state point for each of the three networks examined and the five data points associated with each average message delay curve for a particular network.

Two approaches exist in determining the number of simulation trials necessary to ensure validity of the results obtained. The first approach, a mathematical one, requires that the desired confidence interval be specified which will result in the determination of the number of simulation runs necessary to obtain this interval. The second approach is to choose the number of simulation runs to make and allow the confidence interval to directly result from this choice. The latter approach is chosen for this investigation. Three independent simulation runs are made for each data point used to construct the average message delay curve for a particular network model. Resulting from these three simulation runs are a set of three average message delay values which are to be used in calculating the mean message delay for a particular load that is placed on the network. The standard deviation and variance is then calculated from the mean value. Using three simulation runs per data point, the variance from the mean message delay proves to be reasonably small. For lightly loaded networks (i.e., negligible network queueing), the variance from the mean is less than 2% while for heavily loaded networks, the maximum variance is 9% from the mean. These values indicate, with a high degree of confidence, that the mean message delay values obtained accurately represent the model simulated. These mean message delay values are used for comparison against previously published works to determine the validity of the network models.

5.3 Network Model Validation

As with any modeling effort, a major concern is determining the validity of the model. Two methods exist: compare simulation results with analytic models; or compare simulation results against previously published works. In validating the multistage cube network and single stage cube network models, the latter approach is chosen.

Ideally, when comparing one's models with previously published works, the operating environments of the networks compared should be equivalent to one another.

This gives the basis for accurate comparisons without having to translate network parameters which may differ. Unfortunately, a direct comparison does not exist for the results obtained from the networks modeled for this investigation and the results obtained in previously published works. The major difference in modeling techniques lie in the manner in which messages are generated and presented to the network.

In the works of [DiJ81, DaS86, AbP86, HsY87], a discrete time message generation rate is assumed. By this, the probability of a processor generating a message in a given cycle directly correlates to the message generation rate. If a PE's input buffer to the network is empty, then the PE generates a message with probability m_g [DiJ81]. For example, a generation rate of 100% corresponds to each PE generating a new message packet every time cycle its input buffer is detected to be empty. Controlling the probability of message generation thus controls the load which is placed on the network.

An alternative approach to defining the message generation and network saturation rates in terms of continuous time generation processes is implemented in this investigation. Using an infinite buffer model, the restriction is removed of having a processor's network input buffer empty before the generation of a new message is allowed. This aspect of the model enables the effects of network congestion to be more accurately reflected in the operation of the overall system. Previous work is based on the observation that a heavily loaded network does not empty all of its input buffers immediately and, as a result, "cut-off" the arrival of new messages. The time that the inputs are cut-off is not included in the published packet delay statistics. The approach taken in this investigation serves to decouple the state of the network and message generations. Therefore, packet delay statistics for heavily loaded networks reflect both normal network queueing delays *and* the delay incurred waiting to enter the network (a measure of the time the PE is idled in previous work). Note that, in either approach, the network congestion and the generation process are effectively decoupled under light loading conditions when the network's input buffers would rarely block the arrival of new messages. In the network models created for this investigation, any message that is generated is queued in the input buffer associated with the generating PE if blockage occurs or is allowed to proceed to the next network buffer in route toward its specified destination PE.

To make an accurate comparison of the average message delay times produced in this investigation to the delay times presented in previously published works, a correlation of the message generation rates used is needed. This correlation proves to be an approximation due to the subtlities of modeling differences in the networks in [DiJ81, KrS83, AbP86, DaS86, HsY87], as discussed above. In translating the message generation rates used in this investigation to closely match the rates of the above named works, it is first necessary to compare the message delay curves generated "in-house" to the delay curves presented in the previously published works. By overlaying the curves of the respective networks, and observing the delay curve trends, an approximate correlation of the message generation rates is established. As a result, the approximate correlation is given as follows: for a discrete probability of generating a message, m_g, where m_g=0.8, the corresponding message generation rate used in this investigation is approximately 2N/3 messages/cycle, with N being the number of PEs in the network; on the other end of the loading scale, where queueing is negligible, an m_g=0.2 is approximately equivalent to N/8 messages/cycle.

5.3.1 Multistage Cube Network Validation Using the above correlation, a comparison of the average message delay times for a 64-PE multistage cube network built with 2-by-2 crossbar switching elements serves as the validation base for a multistage cube network of arbitrary N and crossbar switching element size. The work of [KrS83] is chosen for the validation comparisons. Comparing the average message delay values produced by the simulator to those of [KrS83], the average message delay times differ by at most 6% in the case of a heavily loaded network. Further validity of the model is gained by maximum buffer size comparisons with [DiJ81]. For network loads corresponding to normal processor operating ranges (i.e., an aggregate message generation rate which ensures that the ratio of message service rate to message generation rate does not cause network saturation), the maximum network buffer sizes never exceed 6 packets in length. This implies, and is shown in [DiJ81], that infinite buffer length models can be used to model networks comprised of finite length buffers of small size. Deeming the 64-PE multistage cube network model valid, models for $N > 64$ and crossbar switching element sizes greater than 2-by-2 can also be deemed valid. This validation step is possible due to the manner in which the network is modeled. Any extension in size of the network or its constituent crossbar switching element sizes requires that only size parameters be changed and passed to SLAM. One Fortran module is used to model the operation of a crossbar switching element. A change to the size of a crossbar switching element results in a change in the number of input buffers to the switch that must be examined. The buffer scanning mechanism is logically identical for an arbitrary sized crossbar. Since the size parameter variations only require extensions to modules validated for 2-by-2 switching element sizes, network models constructed using higher order switch sizes may also be deemed valid.

5.3.2 The Single Stage Cube Network Validation The validation of the single stage cube network uses portions of the results obtained in validating the multistage cube network and also uses the work of [HsY87] as its comparative base. In the three interconnection networks modeled, each uses crossbar switching elements to interconnect the system PEs. Based on the validation of the crossbar switching element module, and the message generation correlations discussed above, direct comparisons of the delay curves produced by the simulators are made with those of [HsY87]. As with the multistage cube network, the validation base network size is 64 PEs. Comparing the average message delay times for $0.2 \leq m_g < 0.8$, the simulation results are approximately equivalent to those of [HsY87]. At $m_g \geq 0.8$, the differences between the delay times are approximately 10%. Comparisons of the respective studies for N=256 and N=1024 show that the average message delay times are approximately equivalent up to m_g=0.7. For m_g values above 0.7, the differences are once again approximately 10%. These delay value differences can be attributed to the difference in message generation approaches of [HsY87] and the one used in this investigation as discussed previously.

5.3.3 The Illiac IV Network Validation The validation of the Illiac IV network extends from the validation of the two previously discussed models. As in the multistage cube and single stage cube networks, the Illiac IV network uses crossbar switching elements for inter-PE connections. The network models only differ in the interconnection of the PEs and the routing algorithm used in message passing. Verification of the correctness of the crossbar switch implementation was presented above. The verification of the PE interconnection and shortest path routing results from extensive testing of the interconnection functions and the routing algorithm. Fortran programs serve as the base of the interconnection functions and routing algorithm testing. Iterative testing, in Fortran, using different source-destination address pairs, provides a valid initial approach in determining the accuracy of the interconnection functions and the routing algorithm. A second testing method, the SLAM TRACE construct, provides additional verification of the model.

5.4 Individual Network Performance

In modeling the interconnection networks, three network sizes are chosen for implementation, N = 64, N = 256, and N = 1024. These network sizes are representative of systems which are technologically implementable using current microprocessor technology. In each of the multistage cube models, a choice exists for the size of crossbar switching element to be used in the network implementation. In the case of a 64-PE network, switch sizes of 2-by-2, 4-by-4, and 8-by-8 can be used in modeling the network. Multistage cube networks of size N = 256 and N = 1024 can also use multiple switch sizes in their respective implementations. For N = 256, the possible sizes are: 2-by-2, 4-by-4, and 16-by-16 while for N = 1024, 2-by-2, 4-by-4, and 32-by-32 size switches can be used. Switch sizes greater than 32-by-32 are not technologically feasible at the present (due to integrated-circuit pin-out limitations) and are therefore not implemented for simulation. By varying the switch sizes, the performance characteristics resulting from these variations can be observed. For N = 256, and N = 1024, crossbar switch sizes of 2-by-2 are excluded from implementation and evaluation due to the fact that the minimum message delays associated with each of the implementations are greater than those associated with the mesh and single stage cube networks of equal size.

Figure 5.1 shows the family of average message delay curves associated with a 64-PE multistage cube network. For comparative purposes, the delay curve for a 64-by-64 crossbar switch is included. Evident from this figure is the reduction in average message delay times as the size of the crossbar switch is increased. This is due in large part to reduction in the number of stages that must be traversed by a message as the crossbar switch size is increased. Also, as the crossbar switch size is increased, the aggregate message arrival rate must be increased to drive the network into saturation. This trend is in agreement with previous studies [Pat81, KrS83], that have examined the effects of larger size crossbar switches. Similar effects can be observed for larger size networks of size N = 256, and N = 1024. Figures 5.2 and 5.3 display the average message delay curves for these larger networks.

Besides the differences in interconnection functions and routing schemes, single stage cube network implementations differ from multistage stage cube network implementations in that the crossbar switching element size is fixed for a chosen network size. Recall from Chapter 2, that for a m-dimension single stage cube network, the crossbar switching elements that are used for inter-PE connections are of size $(m+1)$-by-$(m+1)$. As an example, consider an 8-dimension single stage cube which consists of $2^8 = 256$ PEs and 256 crossbar switches. The size of the crossbar switch needed for construction of this network is 9-by-9.

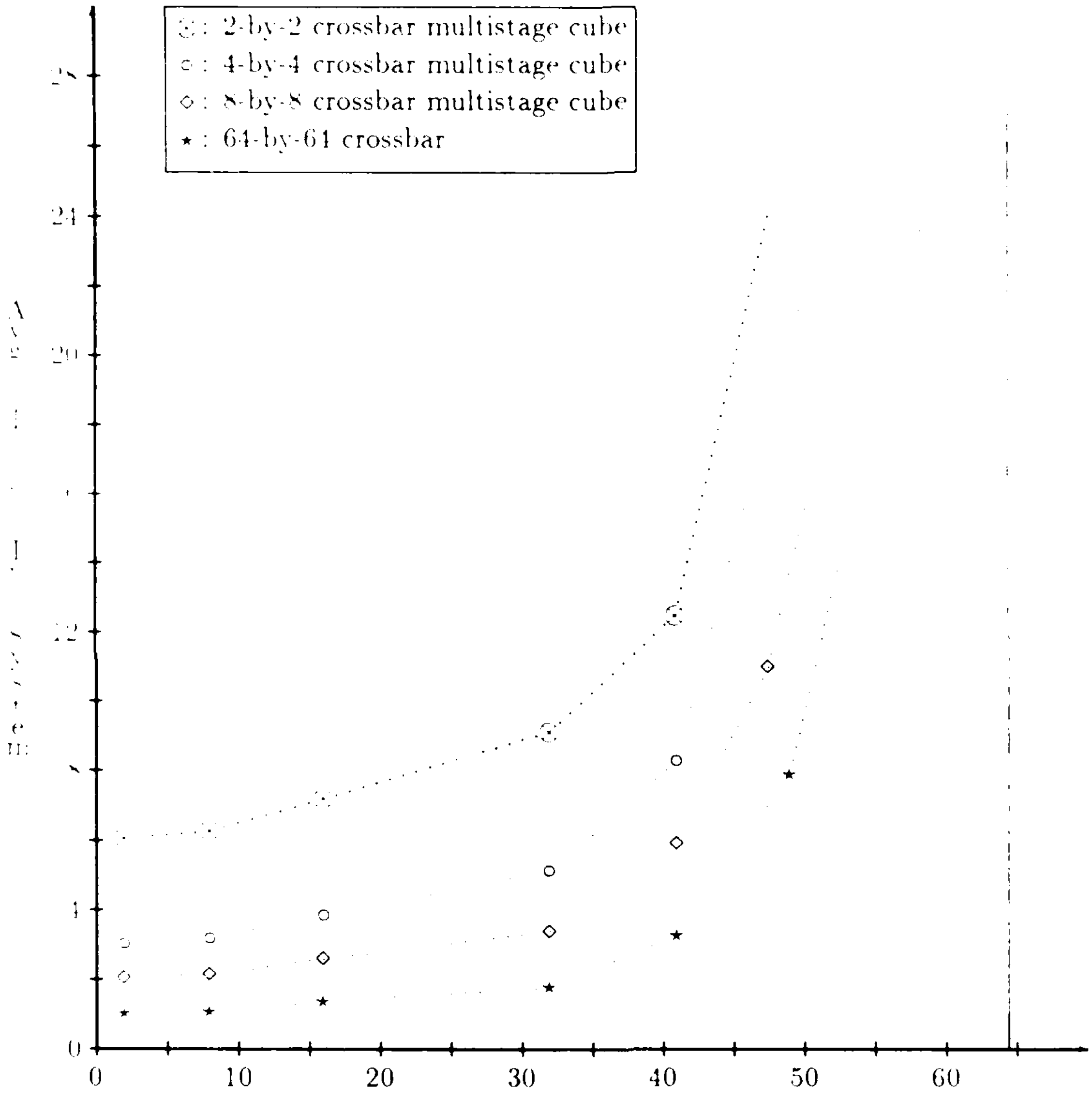


Figure 5.1. Average message delay vs message arrival rate for a 64-PE multistage cube network implementations.

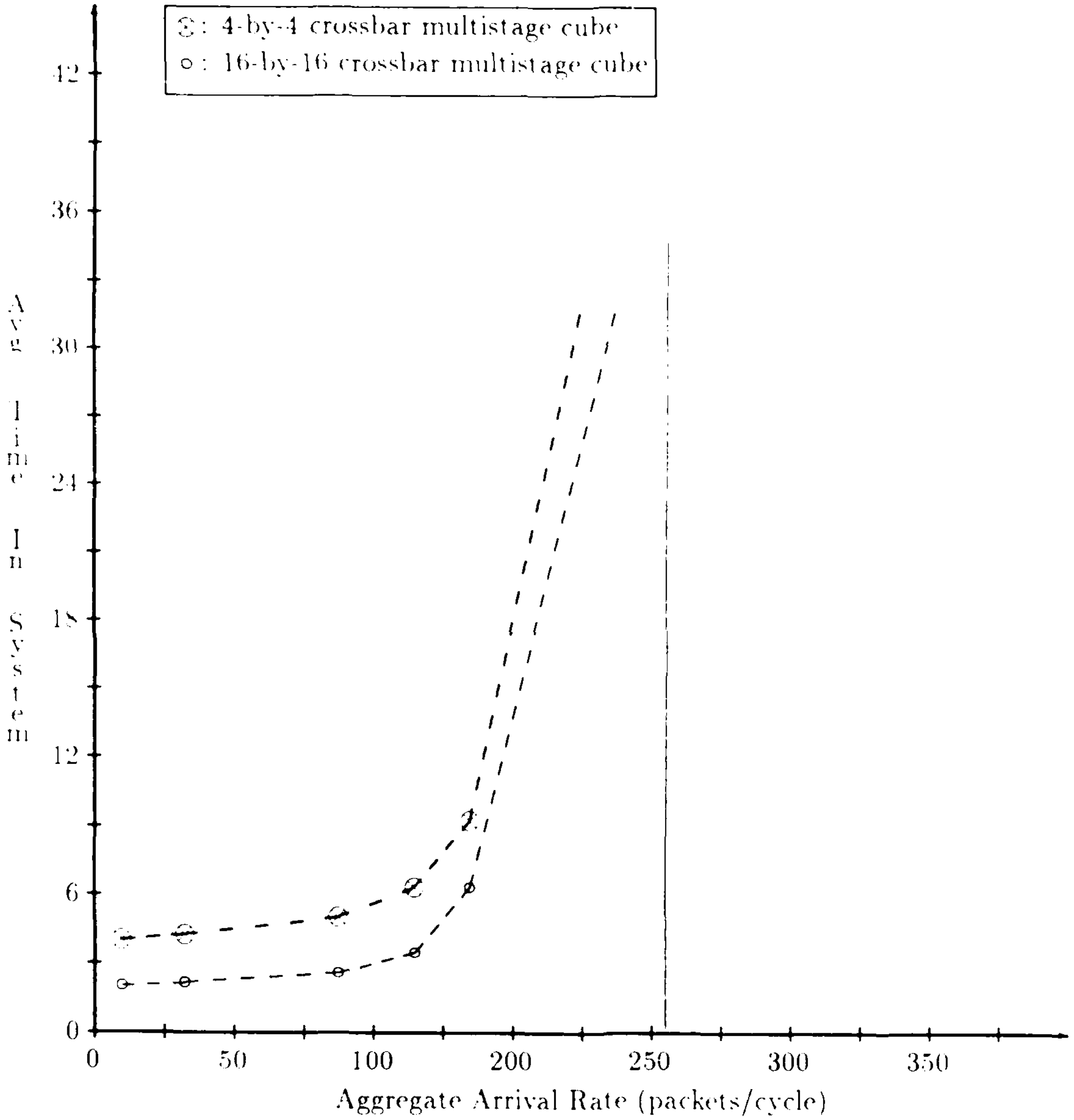


Figure 5.2. Average message delay versus message arrival rate for 256-PE multistage cube network implementations.

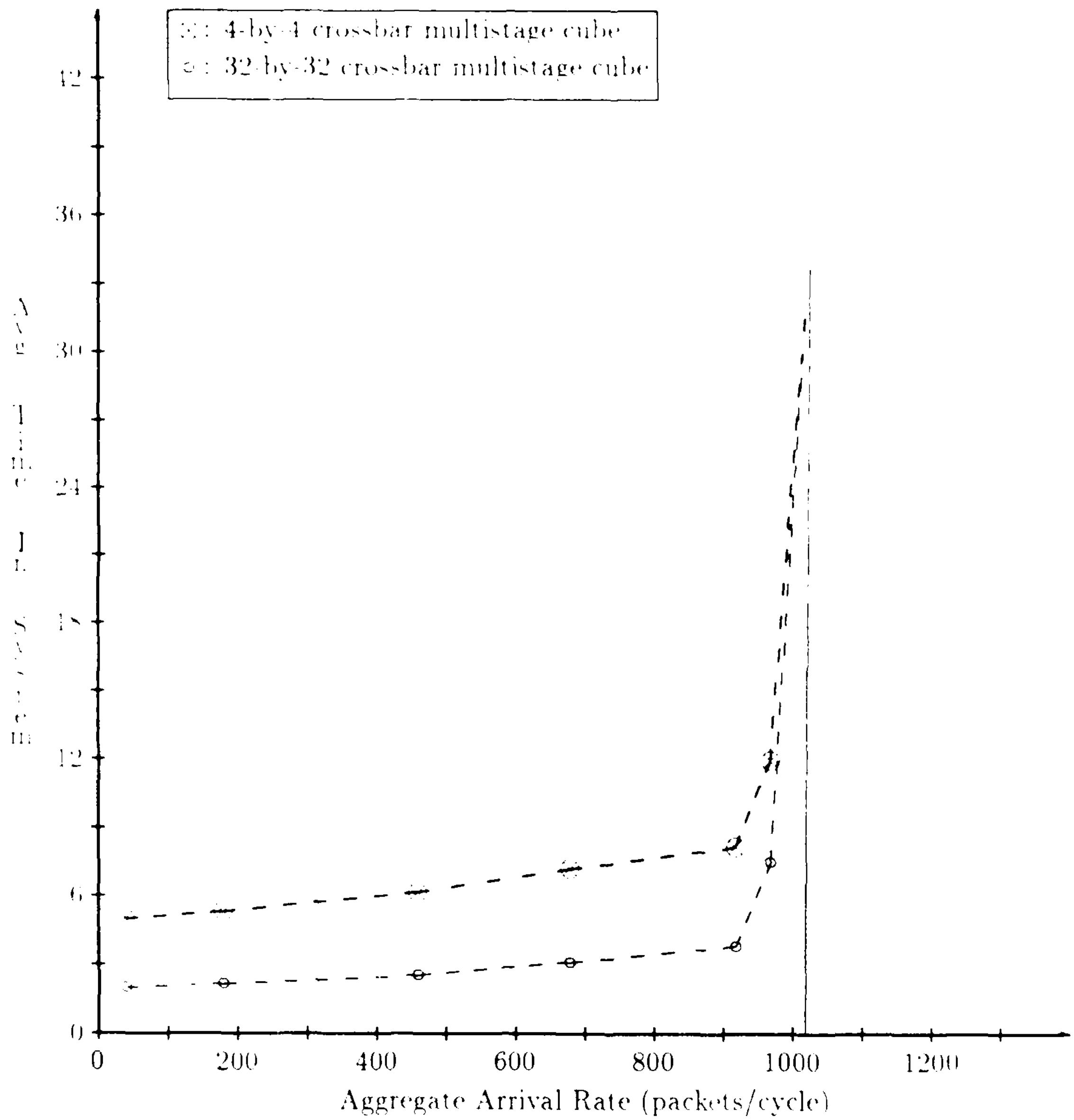


Figure 5.6. Average message delay versus message arrival rate for 1024-PE multistage cube implementations.

Since only one implementation exists for a given network size, the single stage cube network performance is discussed in the network comparisons which follow.

In the construction of an Illiac IV network, the crossbar switching element size is fixed at 5-by-5 for any allowable network size. Network sizes may be the square of any integer value greater than 1. For the purposes of comparison against the other two networks modeled, the Illiac IV network is modeled for network sizes of 8^2, 16^2, and 32^2. As in the case of the single stage cube model, only one implementation of the Illiac IV network exists for a given network size. Therefore, the performance discussion of the Illiac IV network is presented in the section which follows.

5.5 Network Performance Comparisons

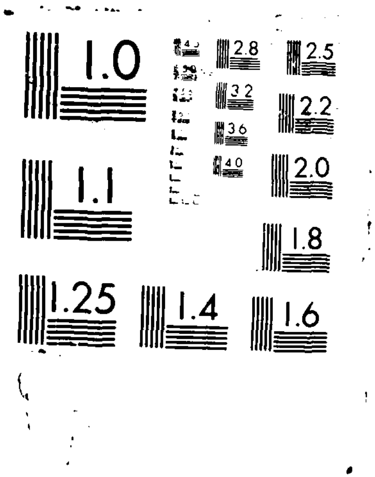
In analyzing the results obtained from the three network simulations, certain trends appear constant as the sizes of the networks are varied. These trends, discussed below, allow for the discussion on the comparison and analysis of the three networks to be focussed on one network size, where N = 256.

Before a comparison of the three network models' message delay characteristics can be made, it is necessary to understand how the networks differ in the minimal obtainable message delays. Using a fixed message cycle time (the packet cycle time), which includes the processing time internal to the crossbar switch, the minimal number of packet cycles that it takes a message to traverse the network is quantifiable. In the multistage cube network, the number of hops from the source PE to the destination PE is dependent upon the number of stages in the network, and is explicitly determined by the size of the crossbar switching element used in the network. Consider a network supporting 256 PEs. For the 4-by-4 switching element implementation, 4 hops are required to traverse the network with a minimum delay of 4 packet cycles (the delay will increase as a result of queueing actions within the network caused by an increase in network load). For a 16-by-16 implementation, 2 hops and a delay of at least 2 cycles will be incurred by a message traversing the network. The minimum number of hops and packet cycles messages incur in network traversals becomes variable in the single stage cube and Illiac IV networks. The minimum number of hops and packet cycles a message requires in the single stage cube and the Illiac IV networks are 1 and 2 respectively. PEs that are directly connected (via respective interconnection functions) are considered one hop distance away from one another but require a minimum of two packet cycles to move through the crossbar switching elements associated with each PE. The maximum number of hops required to route a message in the single stage cube network is dependent upon the dimension of the network. For N = 256, the dimension of the cube is 8, indicating that the worst case distance that must be traveled by a message is 8 hops. The minimum number of packet cycles required for the worst case distance is 9, which is equal to the network dimension plus one. Similarly, in the Illiac IV network, the worst case hop distance is $\sqrt{N}$ and the minimum number of packet cycles required for this worst case is $\sqrt{N} + 1$. In both cases, N is the number of PEs in the network.

Shown in Figures 5.4, 5.5, and 5.6 are the average message delay curves for networks of size N = 64, N = 256, and N = 1024 at various loading levels. Figure 5.5 is used for the discussion of the trends alluded to above. Note that for relatively light loading (i.e., the aggregate message arrival rate of 2 packets/cycle), the average message delay for the single stage cube network and the Illiac IV network are the average of the sum of the minimum and maximum hop times, 5 and 9 respectively. This is the lowest average delay time possible for message assigned so[illegible] addresses from an uniform distribution.

Evident from Figure 5.5 is the preference of the [illegible] two networks for light to medium loading. At an aggre[illegible] of N/2 packets/cycle, the single stage cube [illegible] stage cube network constructed with [illegible] characteristics associated with ea[illegible] in the saturation points of the [illegible]

AD-A190 571 THE MODELING SIMULATION AND COMPARISON OF 2/2
INTERCONNECTION NETWORKS FOR PA..(U) AIR FORCE INST OF
TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI..
UNCLASSIFIED R A RAINES DEC 87 AFIT/GCE/ENG/87D-8 F/G 25/5 NL


1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

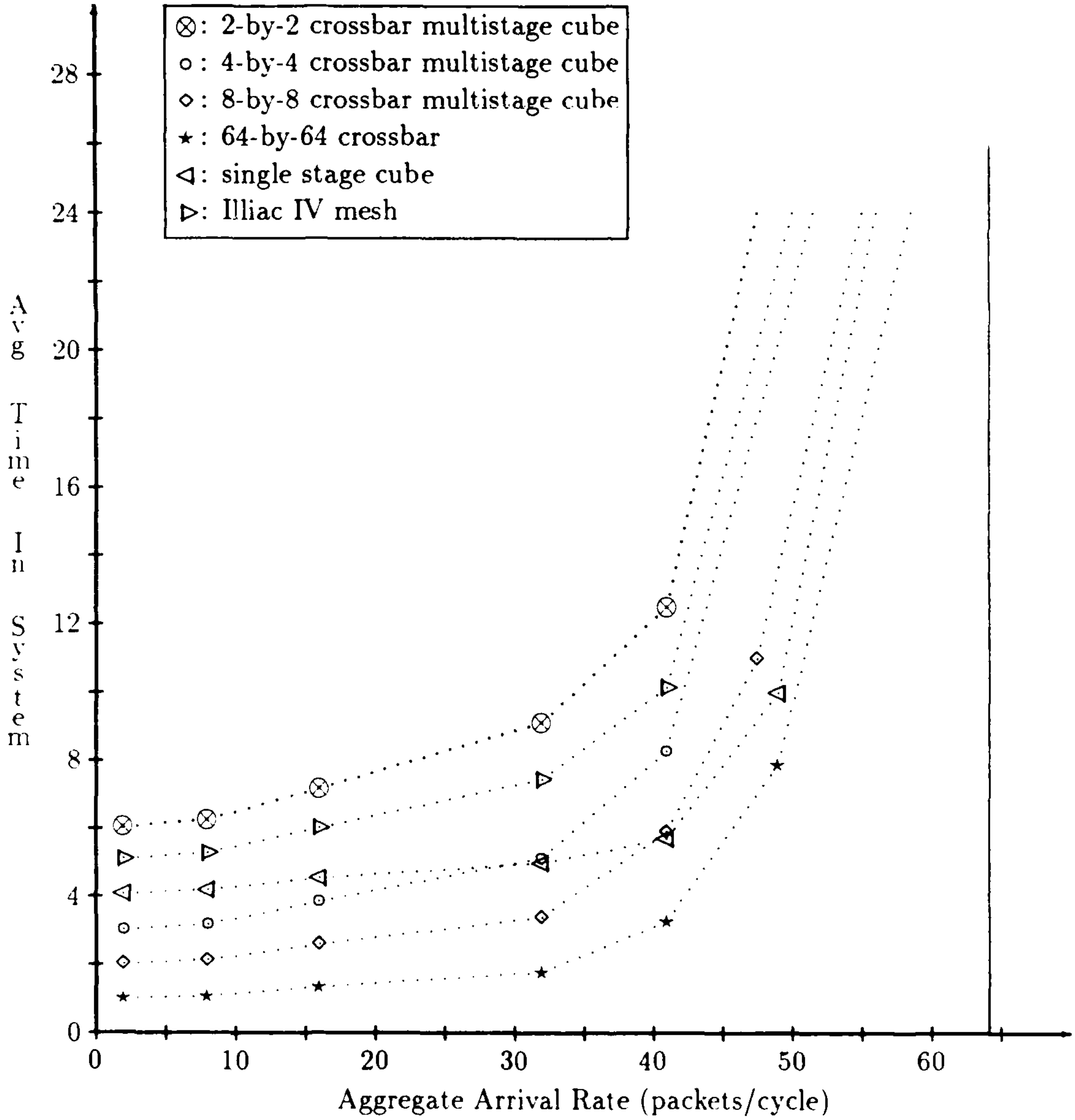


Figure 5.4. Average message delay vs message arrival rate for a 64-PE networks.

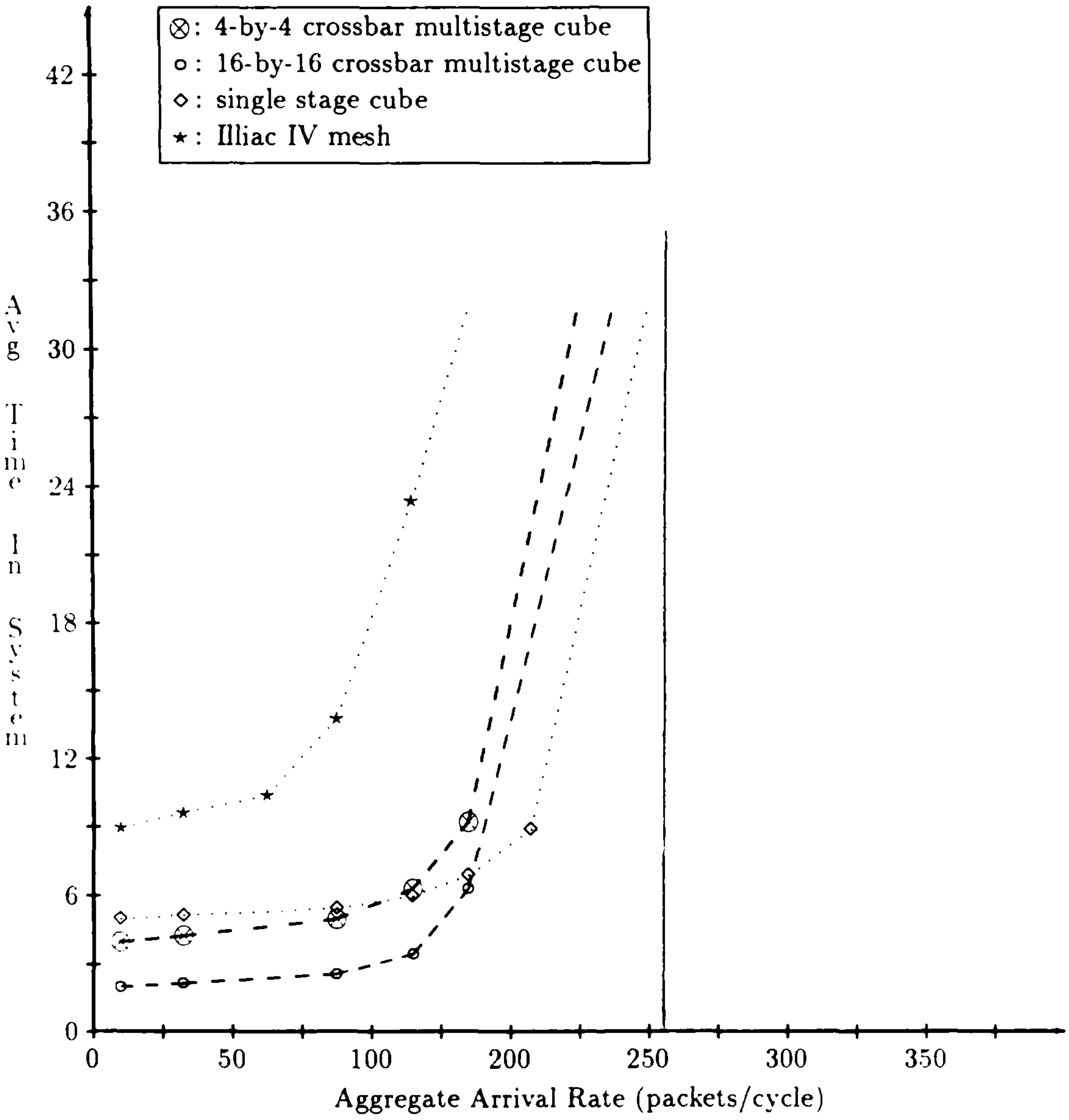


Figure 5.5. Average message delay versus message arrival rate for 256-PE networks.

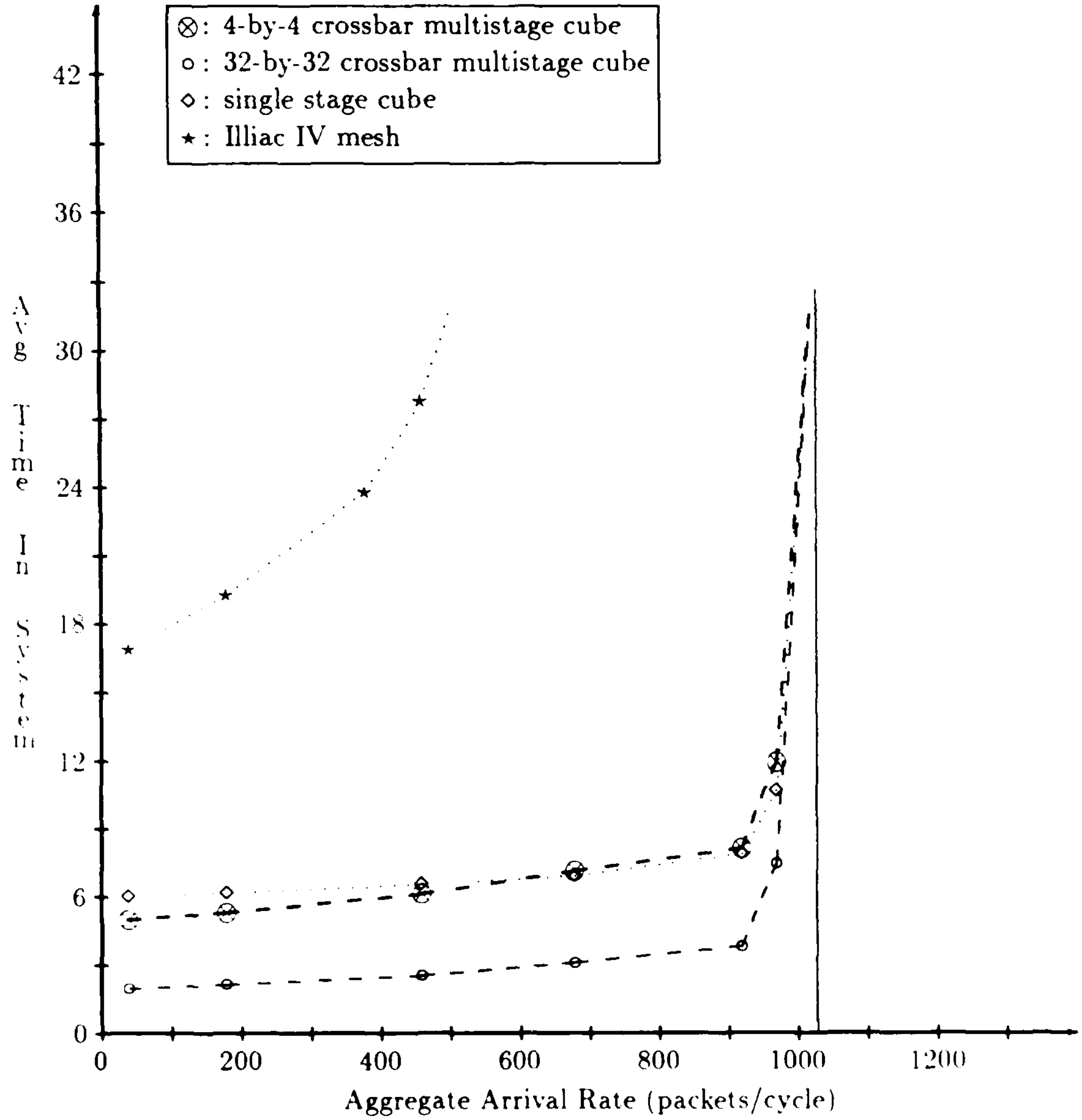


Figure 5.6. Average message delay versus message arrival rate for 1024-PE networks.

the sizes evaluated. As the aggregate message arrival rate is increased, the networks tend to saturate in the following order: the Illiac IV network, the multistage cube network, and then the single stage cube network. Saturation occurs first in the Illiac IV network due primarily to the greater average hop distance that messages must travel in traversing the network from source PE to destination PE. For a given message arrival rate, the larger hop distances experienced in the Illiac IV cause more message to be in the network at any instance of time. This, in turn, causes increases in message congestion resulting in increases in network queueing and greater average message delay times. In the 64-PE and 256-PE networks, the average hop distances that message travel are approximately double the hop distances of the single stage cube network and in certain cases, four times the hop distances required in the multistage cube network. This disparity in average hop distances becomes more profound as the network sizes increase. As a result, the nonsaturation operating range of the Illiac IV network is severely limited in comparison to the single stage cube and the multistage cube networks.

The saturation characteristics evident in the single stage cube networks result directly from the average message hop distances and the number of packet buffers required for the network implementation. While the average message hop distances closely compare with those of the multistage cube networks, the number of packet buffers required to implement a single stage cube can be as much as five times (for N = 1024), the number required to implement a multistage cube network of similar size. In the case of a 256-PE single stage cube network, the number of packet buffers required is a two-fold increase over the 256-PE multistage cube network constructed with 4-by-4 crossbar switching elements, and a four-fold increase over a 256-PE multistage cube network built with 16-by-16 crossbar switching elements. This causes the single stage cube network to have less message congestion for uniform network loading than the multistage cube network of similar size, for a given aggregate message arrival rate. Less message congestion results in less network queueing delays and a greater nonsaturated network operating range.

Comparisons of the single stage cube delay characteristics with those of the multistage cube network, constructed with 16-by-16 crossbar switching elements, show that the multistage cube network performs better than the single stage cube network for light to moderately heavy network loading. This is due to the multistage cube's reduced minimum hop distances (2), which requires a heavier network loading to cause average message delays to approximate those of the single stage cube network. Only when the aggregate message generation rate is greater than 2N/3 are the the delay characteristics approximately equivalent for the single stage cube and the multistage cube network built with 16-by-16 crossbar switching elements. As the message load increases, the message queueing becomes more predominant in the multistage cube network and as a result, the average message delay times become larger than those in the single stage cube network for a given load.

From the discussion above, conclusions can be made concerning the implementation preference of one interconnection network over another based entirely on the average message delay characteristics associated with each network. The physical structure of the Illiac IV network leads to inherently large average message delay times and restricted operating ranges (network loads). If these are the dominant performance factors in a network's design, the mesh network is not the best choice of network topologies for implementation given the three investigated topologies. A choice between the multistage cube network and the single stage cube network lies ultimately in the size of the network and the size of the crossbar switching element to be used in the implementation of the multistage cube network. Recall from Chapter 3, [AbP86] showed that for N = 4096, crossbar switching elements of 8-by-8 and larger must be implemented in the multistage cube network for it to outperform (in terms of average message delay), the single stage cube network. This investigation provides similar results where a reduction in the size of the network reduces the crossbar switching element sizes of the multistage cube network necessary to outperform the single stage cube network.

The second performance parameter to be discussed in this investigation is the memory cost associated with implementing a chosen network. Tables 5.1, 5.2, and 5.3 show the hardware breakdown associated with each network and the maximum packet buffer length requirements for a given aggregate message arrival rate. Recall from Section 5.2, that the maximum packet buffer lengths shown in the tables listed above, are the lengths necessary to ensure that 99% of the time that the network is in operation, these length will not be exceeded. Referring to Tables 5.1, 5.2, and 5.3 certain trends can be observed. Note the maximum packet buffer lengths of the single stage cube network. For the aggregate message arrival rates chosen, the buffer sizes are equivalent across the network sizes evaluated for a given network load. Comparing the maximum buffer sizes of the single stage cube network at increased network loading to the sizes required in the multistage cube and Illiac IV networks shows that the single stage cube network requires buffer lengths that are approximately 1/2 to 1/5 less than the requirements for the multistage cube and Illiac IV networks respectively.

This difference in buffer sizes is due to the large disparity between the number of packet buffers that exist in each of the networks. As an example, consider the 256-PE networks. The buffer requirements of the multistage cube network are 512 and 1024 buffers for implementations using 16-by-16 and 4-by-4 crossbar switching elements respectively. The Illiac IV requires 1280 buffers while the single stage cube needs 2304 buffers for its construction. Having two-fold and four-fold increases packet buffers over the multistage cube networks of 4-by-4 crossbars and 16-by-16 crossbars, allows the messages entering the single stage cube network to be more widely distributed causing less message congestion for uniform network loading. The reduction in packet buffer length requirements in the single stage cube network is not enough to make the single stage cube the most economical network to implement. Tables 5.4, 5.5,

Network	Crossbar switch size	Buffers in network	Aggregate message arrival rate (pkts/cycle)				
			2	N/8	N/3	N/2	2N/3
			Maximum buffer size (pkts)				
multistage cube	2-by-2	384	1	2	4	6	9
multistage cube	4-by-4	192	1	2	4	6	11
multistage cube	8-by-8	128	1	2	4	6	10
crossbar	64-by-64	64	1	2	4	6	11
single stage cube	7-by-7	448	1	2	3	3	4
Illiac IV	5-by-5	320	1	2	3	4	7

Table 5.1. Buffer requirements for 64-PE networks to avoid overflow 99% of time.

Network	Crossbar switch size	Buffers in network	Aggregate message arrival rate (pkts/cycle)				
			2	N/8	N/3	N/2	2N/3
			Maximum buffer size (pkts)				
multistage cube	4-by-4	1024	1	2	3	6	9
multistage cube	16-by-16	512	1	2	4	6	10
single stage cube	9-by-9	2304	1	2	3	3	4
Illiac IV	5-by-5	1280	1	2	6	13	22

Table 5.2. Buffer requirements for 256-PE networks to avoid overflow 99% of time.

Network	Crossbar switch size	Buffers in network	Aggregate message arrival rate (pkts/cycle)				
			2	N/8	N/3	N/2	2N/3
			Maximum buffer size (pkts)				
multistage cube	4-by-4	5120	1	2	3	4	6
multistage cube	32-by-32	2048	1	2	3	5	7
single stage cube	11-by-11	11264	1	2	3	3	4
Illiac IV	5-by-5	5120	1	2	10	17	23

Table 5.3. Buffer requirements for 1024-PE networks to avoid overflow 99% of time.

and 5.6 shows the unit cost versus network load comparison for each of the network sizes investigated. As defined in Section 5.2, the cost of a network is the product of the number of buffers in the network, the maximum buffer length requirement and the unit dollar cost per specific buffer memory size. This cost estimate also assumes that the dollar costs differences of implementing one size crossbar switching element over another are negligible. As a clarifying example, consider the networks of size N = 256. At light network loading, each network requires that the maximum buffer length be one packet. The network with the least cost requirement (512 cost units) is the multistage cube network constructed from 16-by-16 crossbar switching elements. At heavy network loading, the 16-by-16 crossbar implementation of the multistage cube still provides for the lowest cost requirement (5120 cost units), but now the cost of implementing a single stage cube network (9216 cost units) is equivalent to the cost of a multistage cube network constructed using 4-by-4 crossbar switching elements. Only at loads greater than 2N/3 packets/cycle is the single stage cube network's total memory cost approximately the same as the multistage cube network's cost. The total memory costs for the Illiac IV network are only comparable for light network loading due to the increased queueing delays the network experiences.

Network	Crossbar switch size	Aggregate message arrival rate (pkts/cycle)				
		2	N/8	N/3	N/2	2N/3
		Network cost (cost units)				
multistage cube	2-by-2	384	768	1536	2304	3456
multistage cube	4-by-4	192	384	768	1152	2112
multistage cube	8-by-8	128	256	512	768	1280
crossbar	64-by-64	64	128	320	448	704
single stage cube	7-by-7	448	896	1344	1344	1792
Illiac IV	5-by-5	320	640	960	1280	2240

Table 5.4. 64-PE network cost per load comparison.

Network	Crossbar switch size	Aggregate message arrival rate (pkts/cycle)				
		2	N/8	N/3	N/2	2N/3
		Network cost (cost units)				
multistage cube	4-by-4	1024	2048	3072	6144	9216
multistage cube	16-by-16	512	1024	2048	3072	5120
single stage cube	9-by-9	2304	4608	6912	6912	9216
Illiac IV	5-by-5	1280	2560	7680	16,640	28,160

Table 5.5. 256-PE network cost per load comparison.

Network	Crossbar switch size	Aggregate message arrival rate (pkts/cycle)				
		2	N/8	N/3	N/2	2N/3
		Network cost (cost units)				
multistage cube	4-by-4	5120	5120	15360	20480	30720
multistage cube	32-by-32	2048	4096	6144	10240	14336
single stage cube	11-by-11	11264	22528	33792	33792	45056
Illiac IV	5-by-5	5120	10240	51200	87040	117760

Table 5.6. 1024-PE network cost per load comparison.

5.6 *Summary*

In this chapter, the simulation, validation, and performance comparisons of the multistage cube, the single stage cube, and the Illiac IV interconnection networks have been presented. An in-depth discussion of the simulation tools and techniques provided information about the machine requirements necessary for large scale simulations and methods used to ensure the correctness of the simulations. Section 5.3 presented an overview of validation techniques to include the validation of the three interconnection network models developed for this investigation. Individual network performance characteristics were presented in Section 5.4 followed by a performance comparison of the network models. From this comparison, it was observed that due to its physical structure, the Illiac IV network has inherently large message delay times and restricted nonsaturation operating ranges as compared to the single stage cube anu multistage cube networks. It was also observed that, the implementation choice of single stage cube network or multistage cube network is dependent upon the network size to be implemented and the size of the crossbar switching element used in constructing the multistage cube network.

6. Conclusions and Recommendations

6.1 Summary of Thesis Investigation

Before addressing the conclusions and recommendations resulting from this investigation, a brief review of the material presented in this thesis effect is necessary. In Chapter 1, a background review of restrictions and limitations of the traditional von Neumann computer was presented. The research goals of this investigation were defined to allow for an accurate scope of the problem at hand.

Chapters 2 and 3 provided the background information necessary understand the techniques used in parallel processing system evaluation. Chapter 2 defined the methodologies used in classifying parallel processing systems and the problems inherent with each methodology. Interconnection network topologies were introduced in Chapter 2 followed by discussions of contemporary parallel processing systems which used these interconnection network implementations. Chapter 3 examined the state of parallel processing systems evaluations to include crossbar switch analysis, network switching methodology analysis, and network performance comparisons.

The methodology used in solving this investigation was the topic of Chapter 4. In this chapter, the simulation tool used to model the three interconnection networks was discussed. The formulation of network models to include interconnection functions and routing algorithms were presented to provide insight into the "inner workings" of the simulator. Chapter 5 began with a discussion of the network simulation methodology. This discussion included the facilities necessary to perform the investigation and the manner in which the simulations were performed. Network validation procedures were used to ascertain the validity of the networks models as compared to previously published works. The performance measures of average message delay, maximum packet buffer lengths, and total network implementation costs provided for the conclusion that the decision of implementing a single stage cube network or a multistage cube network requires forethought into the size of the network to be implemented and the size of the crossbar switching element to be allowed for implementation of the multistage cube network. Only when these two factors have been determined can a clear-cut choice be made. The results obtained in Chapter 5 further indicate that the Illiac IV network's physical structure makes it the least desirable of the networks investigation when considering the average message delay as the dominant performance factor.

6.2 Thesis Effort Conclusions

Many points can be concluded from this thesis investigation. First, this investigation has provided a unified base for the comparison of three classes of interconnection networks, which to the present, has not been done. A second point is the alternative approach to viewing the average message delay and network saturation rates. This alternative approach has provided for a more accurate method of determining the delay characteristics of a particular network. Thirdly, this investigation has shown that large-scale simulations of parallel processing systems are feasible using a commercially packaged simulation language such as SLAM. And lastly, an alternative method of SLAM file scanning has been developed to add flexibility to the language.

6.3 Recommendations for Future Research

This investigation has provided a comparison of three classes of interconnection networks under a common set of system operating assumptions which had not been previously performed. Due to the diversity, complexity, and time restraints of this investigation, certain simulator enhancements could not be implemented. These enhancements to the simulator form a base for future research in the area of interconnection network performance comparisons. These proposed enhancements are as follows:

1. Implement dynamic routing for the single stage cube and Illiac IV networks. By doing so, the inherent message delay characteristics of the Illiac IV network may be changed and make this network more desirable for implementation.
2. Simulate the effects of nonuniform loading on the networks. This will allow for the determination areas of network congestion and provide insight to possible method for removing the congestion.
3. Implement a model which allows for the partitioning of the multistage cube network.

In closing, this investigation has shown that commercial simulation packages such as SLAM can be used for parallel processing applications. Using simulation languages allow for compact coding and ease of readability which result in a faster design and implementation turnaround. This thesis further serves as a link which has previously been missing in the performance comparisons of classes of interconnection networks. As a result, two technical papers, [RaD88a], and [RaD88b], have been submitted for publication.

Appendix A. *Source Code Release Information*

Further information concerning the SLAM and Fortran source code developed for this investigation may be obtained by contacting Captain Nathaniel J. Davis IV or Captain Wade H. Shaw in the Department of Electrical and Computer Engineering, Air Force Institute of Technology, Wright-Patterson AFB, OH 45433.

Bibliography

AbP86. S. Abraham and K. Padmanabhan, "Performance of the direct binary n-cube network for multiprocessors," *1986 International Conference on Parallel Processing*, August 1986, pp. 636-639.

AdS84. G. B. Adams III and H. J. Siegel,"The use of 4x4 switching elements in the multistage cube network," *First International Conference on Computers and Applications*, June 1984.

AlH86. N. Al-Holou, "Interconnection networks of multiprocessor systems: general simulation and performance analysis," Ph.D. Thesis, Graduate Engineering and Research School of Engineering, University of Dayton, July 1986.

BaB68. G. H. Barnes, R. Brown, M. Kato, D. J. Kuck, D. L. Slotnick and R. A. Stokes, "The Illiac IV computer," *IEEE Transactions on Computers*, Vol. C-17, August 1968, pp. 746-757.

Ben65. V. Benes, *Mathematical Theory of Connecting Networks*, Academic Press, N.Y., 1965.

Che82. P-Y Chen, *Multiprocessor Systems: Interconnection Networks, Memory Hierarchy, Modeling, and Simulations*, Report Number UIUCDCS-R-82-1083, Department of Computer Science, University of Illinois at Urbana-Champaign, 1982.

ChH84a. C. Y. Chin and K. Hwang, "Priority queueing analysis of packet switching networks," *1984 International Conference on Parallel Processing*, August 1984, pp. 220-224.

ChH84b. C. Y. Chin and K. Hwang, "Packet switching networks for multiprocessors and data flow computers," *IEEE Transactions on Computers*, Vol. C-33, November 1984, pp. 991-1003.

ChL83. P. Y. Chin, J. E. Lilienkamp and D. H. Lawrie, *Performance of Circuit Switched Multistage Interconnection Networks in Multiprocessing Systems*, Department of Computer Science, University of Illinois at Urbana-Champaign, 1983.

CrG85. W. Crother, J. Goodhue, E. Starr, R. Thomas, W. Milliken and T. Blackadar, "Performance measurements on a 128-node Butterfly parallel processor," *1985 International Conference on Parallel Processing*, August 1985, pp. 531-540.

Dal86. W. J. Dally, "Wire-efficient VLSI multiprocessor communication networks," Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1986.

Dav85. N. J. Davis IV, "Multistage Interconnection Networks: Modeling, Performance Analysis, Design, and Fault Location," Ph.D. Thesis, School of Electrical Engineering, Purdue University, August 1985.

DaS85. N. J. Davis IV and H. J. Siegel, "The performance analysis of partioned circuit switched multistage interconnection networks," *Twelfth Annual Symposium on Computer Architecture*, June 1985, pp. 387-394.

DaS86. N. J. Davis IV and H. J. Siegel, "Performance studies of multiple-packet multistage cube networks and comparison to circuit switching," *1986 International Conference on Parallel Processing*, August 1986, pp. 108-114.

DiJ81a. D. M. Dias and J. R. Jump, "Packet switching interconnection networks for modular systems," *IEEE Computer*, Vol. 14, December 1981, pp. 43-53.

DiJ81b. D. M. Dias and J. R. Jump, "Analysis and simulation of buffered delta networks," *IEEE Transactions on Computers*, Vol. C-30, April 1981, pp. 273-282.

Fen72. T. Y. Feng, "Some characteristics of associative/parallel processing," *1972 Sagamore Computer Conference on Parallel Processing*, 1972, pp. 5-16.

Fen81. T. Y. Feng, "A survey of interconnection networks," *IEEE Transactions on Computers*, December 1981, pp. 12-27.

Fly66. M. J. Flynn, "Very high-speed computing systems," *Proceedings of the IEEE*, Vol. 54, December 1966, pp. 1901-1909.

Gar85. A. B. Garcia, "Estimating computer communication network performance using network simulations," Ph.D. Thesis, School of Engineering, University of Dayton, April 1985.

GoG83. A. Gottlieb, R. Grishman, C. P. Kruskal, K. P. McAuliffe, L. Rudolph and M. Snir, "The NYU Ultracomputer - designing an MIMD shared memory parallel computer," *IEEE Transactions on Computers*, February 1983, pp. 276-290

Hän77. W. Händler, "The impact of classification schemes on computer architectures," *1977 International Conference on Parallel Processing*, August 1977, pp. 7-15.

HsY87. W. T. Hsu, P. C. Yew, and C. Q. Zhu, "An enhancement scheme for hypercube interconnection networks," *1987 International Conference on Parallel Processing*, August 1987, pp. 820-823.

HwB84. K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing*, McGraw-Hill, N.Y., 1984.

Int86. iPSC User's Guide, Intel Corporation, April 1986, pp. 6.14-6.22.

KeK79. P.Kermani and L. Kleinrock, "Virtual Cut-Through: A New Computer Communication Switching Technique," *Computer Networks*, Vol. 3, 1979, pp. 337-353.

KrS83. C. P. Kruskal and M. Snir, "The performance of multistage interconnection networks for multiprocessors," *IEEE Transactions on Computers*, Vol. C-32, December 1983, pp. 1091-1098.

Law75. D. H. Lawrie, "Access and alignment of data in an array processor," *IEEE Transactions on Computers*, Vol. C-24, December 1975, pp. 1145-1155.

LeW84. M. Lee and C-L Wu, "Performance analysis of circuit switching baseline interconnection networks," *Eleventh Annual Symposium on Computer Architecture*, June 1984, pp. 82-90.

McS80. R. J. McMillen and H. J. Siegel, "The hybrid cube network," *Distributed Data Acquisition, Computing, and Control Symposium*, December 1980, pp. 11-22.

McA81. R. J. McMillen, H. J. Siegel and G. B. Adams III, "Performance and implementation of 4x4 switching nodes in an interconnection network for PASM," *1981 International Conference on Parallel Processing*, August 1981, pp. 229-233.

MuM82. T. N. Mudge and B. A. Makrucki, "An approximate queueing model for packet switched multistage interconnection networks," *Third International Conference on Distributed Computing Systems*, October 1982, pp. 556-561.

Ove82. A. L. Overvig, *The Simulation of the Generalized Cube Interconnection Network*, Master's Thesis, School of Electrical Engineering, Purdue University, 1982.

Pat81. J. H. Patel, "Performance of processor-memory interconnections for multiprocessors," *IEEE Transactions on Computers*, Vol. C-30, October 1981, pp. 771-780.

Pea77. M. C. Pease III, "The indirect binary n-cube microprocessor array," *IEEE Transactions on Computers*, Vol. C-26, May 1977, pp. 458-473.

PfB85. G. F. Pfister, W. C. Brantley, D. A. George, S. L. Harvey, W. J. Klienfelder, K. P. McAuliffe, E. Melton, V. A. Norton and J. Weiss, "The IBM research parallel prototype (RP3): introduction and architecture," *1985 International Conference on Parallel Processing*, August 1985, pp. 764-771.

Pri86. A. A. B. Pritsker, *Introduction to Simulation and SLAM II*, Systems Publishing Corporation, West Lafayette, IN, 1986.

Pri87. Consultation conference with Pritsker and Associates, West Lafayette, IN, August 13, 1987.

RaD88a. R. A. Raines and N. J. Davis IV, "A comparison of three interconnection networks for parallel processing," *Fifteenth Annual Symposium on Computer Architecture*, May 1988, (submitted for publication).

RaD88b. R. A. Raines, N. J. Davis IV, and W. H. Shaw, "The modeling, simulation, and comparison of interconnection networks for parallel processing," *1988 Summer Computer Simulation Conference*, July 1988, (submitted for publication).

Sie77. H. J. Siegel, "Analysis techniques for SIMD machine interconnection networks and the effects of processor address masks," *IEEE Transactions on Computers*, Vol. C-26, February 1977, pp. 153-161.

Sie85. H. J. Siegel, *Interconnection Networks for Large-Scale Parallel Processing: Theory and Case Studies*, Lexington Books, Lexington, MA, 1985.

SiH86. H. J. Siegel, W. T. Y. Hsu and M. Jeng, "Interconnection networks: the multistage cube, extra-stage cube, and dynamic redundancy networks," *Citicorp New Frontiers in Computer Architecture Conference*, March 1986, pp. 1-19.

SiS84. H. J. Siegel, T. Schwederski, N. J. Davis IV and J. T. Kuehn, "PASM: a reconfigurable parallel system for image processing," *Computer Architecture News*, September 1984, pp. 7-19.

Sto71. H. S. Stone, "Parallel processing with a perfect shuffle," *IEEE Transactions on Computers*, Vol. C-20, February 1971, pp. 153-161.

Sto77. R. A. Stokes, "Burroughs scientific processor," *High Speed Computer and Algorithm Organization*, Academic Press, New York, 1977, pp. 85-89.

Szy86. T. H. Szymanski, "A VLSI comparison of switch-recursive banyan and crossbar interconnection networks," *1986 International Conference on Parallel Processing*, August 1986, pp.192-199.

Tho80. C. D. Thompson, *A Complexity Theory of VLSI*, Department of Computer Science, Carnegie-Mellon University, Technical Report CMU-CS-80-140, August 1980.

Von46. J. von Neumann, H. H. Goldstine and A. W. Burks, "Preliminary Discussion of the Logical Design of an Electronic Computing Instrument," *U.S. Army Ordnance Department Report*, 1946.

Vita

Lieutenant Richard A. Raines was born January 17, 1958 in Tallahassee, Florida. After graduating from Madison High School, Madison, Florida in 1976, he joined the U.S. Army. While receiving his Advance Individual Training in the Nike Hercules Radar and Computer course at Redstone Arsenal, Alabama, he received the Soldier of the Month Award for the 2nd Student Battalion in May 1977 and was the distinguished graduate of his AIT class. Upon graduation from AIT, he was assigned to the DSP 3/71 ADA, Stuttgart, West Germany where he assumed the duties of a Nike Track Radar and Computer Repairman. After completing his Army tour of duty in December 1980, he attended The Florida State University and graduated Cum Laude with a Bachelor of Science in Electrical Engineering in 1985. He received his commission in December 1985 through the Air Force ROTC program and was subsequently assigned to AFIT/RR as a special assistant to the Director of Admissions. In May 1986, Lieutenant Raines entered the Computer Engineering program at the School of Engineering, Air Force Institute of Technology.

Permanent address: 112 N. E. Shelby Street
Madison, Florida 32340

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1a REPORT SECURITY CLASSIFICATION
Unclassified

1b RESTRICTIVE MARKINGS

2a SECURITY CLASSIFICATION AUTHORITY

2b DECLASSIFICATION / DOWNGRADING SCHEDULE

3 DISTRIBUTION / AVAILABILITY OF REPORT
Approved for public release;
distribution unlimited.

4 PERFORMING ORGANIZATION REPORT NUMBER(S)
AFIT/GCE/ENG/87D-8

5 MONITORING ORGANIZATION REPORT NUMBER(S)

6a NAME OF PERFORMING ORGANIZATION
School of Engineering

6b OFFICE SYMBOL *(If applicable)*
AFIT/ENG

7a NAME OF MONITORING ORGANIZATION

6c ADDRESS *(City, State, and ZIP Code)*
Air Force Institute of Technology
Wright-Patterson AFB, OH 45433-6583

7b ADDRESS *(City, State, and ZIP Code)*

8a NAME OF FUNDING / SPONSORING ORGANIZATION
Strategic Defense Initiative Organization

8b OFFICE SYMBOL *(If applicable)*

9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER

8c ADDRESS *(City, State, and ZIP Code)*

10 SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO	PROJECT NO	TASK NO	WORK UNIT ACCESSION NO

11 TITLE *(Include Security Classification)*
see box 19

12 PERSONAL AUTHOR(S)
Richard A. Raines, B.S.E.E., 2d Lt, USAF

13a TYPE OF REPORT
MS Thesis

13b TIME COVERED
FROM ________ TO ______

14 DATE OF REPORT *(Year, Month, Day)*
1987 December

15 PAGE COUNT
114

16 SUPPLEMENTARY NOTATION

17 COSATI CODES

FIELD	GROUP	SUB-GROUP
12	05	
12	06	

18 SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
parallel processing, interconnection networks, simulations, modeling, computer architecture

19 ABSTRACT *(Continue on reverse if necessary and identify by block number)*

Title: THE MODELING, SIMULATION, AND COMPARISON OF
INTERCONNECTION NETWORKS FOR PARALLEL PROCESSING

Thesis Chairman: Nathaniel J. Davis IV, Captain, USA
Assistant Professor of Electrical Engineering

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT
□ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT □ DTIC USERS

21 ABSTRACT SECURITY CLASSIFICATION

22a NAME OF RESPONSIBLE INDIVIDUAL
Nathaniel J. Davis IV, Captain, USA

22b TELEPHONE *(Include Area Code)*
(513) 255-3576

22c OFFICE SYMBOL
AFIT/ENG

Previous editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

This thesis extends existing modeling, analysis and comparison of interconnection networks for parallel processing systems. Simulation models are developed for the multistage cube network, the single stage cube network (hypercube), and the Illiac IV mesh-type network. They are then used to provide a comparison of three classes of interconnection networks which, until now, has not been performed. These models, implemented using a commercially packaged simulation language provide for compact source code and ease of readability. The networks are modeled under a common set of operating assumptions and system environment. This allows for accurate comparisons of average network packet delays and memory requirements necessary to physically implement the chosen network at a given network operating load. It is concluded that, for the network sizes and operating conditions established, the multistage cube network performs better at a lower hardware cost than do the single stage cube and mesh networks. As a result, the designer of a parallel processing system is given additional insight for choosing an interconnection network which best suites the application needs. This thesis investigation is summarized in [RaD88a], and [RaD88b].

END
DATE
FILMED
4-88
DTIC